UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Gestión telemática de vehículos agrícolas

Autor/es

ALBERTO ÚBEDA CAÑAS

Director/es

JAVIER RICO AZAGRA y MONTSERRAT GIL MARTÍNEZ ,

Facultad

Escuela Técnica Superior de Ingeniería Industrial

Titulación

Grado en Ingeniería Electrónica Industrial y Automática

Departamento

INGENIERÍA ELÉCTRICA

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Gestión telemática de vehículos agrícolas

Alberto Úbeda Cañas

Marzo de 2017

RESUMEN

El presente proyecto surge de la necesidad cada vez mayor de introducir la tecnología en las labores cotidianas. Debido a la tendencia de centralizar la gestión de labores en las grandes empresas en el campo de la agricultura, que poseen grandes extensiones de terreno y flotas de vehículos, se presenta la idea de un sistema para la monitorización de estas flotas.

Este sistema permitirá tener información referente a cada vehículo (situación, velocidad y condiciones de trabajo, etc.) en un servidor web, siendo accesible desde cualquier parte del mundo a través de internet, de manera que será posible una mejora en la gestión de los trabajos, puesto que se podrá organizar la flota de vehículos y trabajadores de una manera más eficiente, suponiendo esto mejoras económicas.

Para la elaboración del proyecto se ha empleado hardware de bajo coste, en concreto el microcontrolador Arduino, y junto con este, dispositivos de comunicaciones CANBUS, GPRS y GPS, con el objetivo de obtener y poder comunicar la información más relevante posible.

A lo largo del documento, se explica claramente el proceso llevado a cabo en el desarrollo de este proyecto, hasta finalmente alcanzar una solución totalmente funcional para el cumplimiento de los objetivos propuestos.

Palabras clave:

IoT, Industria 4.0, vehículos inteligentes, telemática.

Sectores:

Maquinaria agrícola, agricultura, tecnologías de la información y las comunicaciones.

SUMMARY

The present Project rises up from the growing necessity of introducing new technologies in daily tasks. Due to the inclination to centralize the management of works in large companies of the area of the agriculture, which own large fields and fleets of vehicles, an idea of monitoring these fleets is presented.

This system will permit to have information referred to each vehicle (situation, speed and working conditions, etc.) into a web server, being accessible from any place in the world through the internet, so that it will be possible an improvement in the work management, since the fleet of vehicles and workers will can be organized in a more efficient way, meaning better economical conditions.

For the preparation of the project, low cost hardware has been used, in particular an Arduino microcontroller, and along with this, CANBUS, GPRS and GPS communication devices, aiming for getting and communicating the most relevant information possible.

Along this document, the process done to develop this project is clearly explained, till finally, reaching a functional solution which fulfills the proposed objectives.

Keywords:

IoT, Industry 4.0, smart vehicles, telematics.

Sectors:

Farm equipment, agriculture, information and communication technologies.

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Gestión telemática de vehículos agrícolas

DOCUMENTO 1: ÍNDICE GENERAL

Alberto Úbeda Cañas

Marzo de 2017

MEMORIA

ANEXO I: PROGRAMAS

ANEXO II: IMÁGENES DE MONTAJE

PLANOS

PLIEGO DE CONDICIONES

PRESUPUESTO

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Gestión telemática de vehículos agrícolas

DOCUMENTO 2: MEMORIA

Alberto Úbeda Cañas

Marzo de 2017

MEMORIA

Índice de ilustraciones

1.-Introducción

Para la superación de los créditos correspondientes al Trabajo Fin de Grado, y con ello, del título de Grado en Ingeniería Electrónica Industrial y Automática, el alumno D. Alberto Úbeda Cañas realiza el presente documento titulado "Gestión telemática de vehículos agrícolas". Como directores del proyecto figuran D. Javier Rico Azagra y Dña. Montserrat Gil Martínez, cuyas indicaciones han ayudado a la conclusión de los objetivos propuestos.

Este trabajo se ubica bajo diferentes áreas del campo de la ingeniería electrónica, en concreto, Instrumentación Electrónica, Informática y Comunicaciones, Electrónica Analógica, Tecnología Electrónica y de Control y Electrónica Digital.

1.1.-Antecedentes

Con la llegada de las nuevas tecnologías, es raro que exista un campo de trabajo que no haga uso de ellas. El mundo de la agricultura no es una excepción. La llegada de los buses de comunicación industrial, la geolocalización, sensores, etc. a la maquinaria agrícola han hecho de este un campo en constante avance.

Cada vez son más los agricultores que optan por montar en sus vehículos equipos GPS para mejorar la precisión de las labores en el campo, o sistemas electrónicos para el manejo de los diferentes implementos que se pueden acoplar al vehículo. Incluso se utiliza la visión artificial por medio de drones, avionetas o satélites para enviar información sobre un cultivo a un vehículo, y gracias a los sistemas de última generación, el propio vehículo sabrá cómo actuar en cada zona de un terreno.

En el presente proyecto se va a tratar el problema planteado por grandes empresas y cooperativas, que cuentan con enormes flotas de vehículos y grandes extensiones de terreno en lugares muy diferentes. Este problema es el de la geolocalización y monitorización de estas flotas para una mejor gestión de las labores a realizar en el campo.

Este tipo de empresas normalmente cuentan con oficinas centrales, desde donde se coordinan todas las actividades a realizar en el tiempo. Desde aquí también se asigna a cada vehículo y operario una labor a realizar.

La ventaja aportada por el presente proyecto radica en la monitorización en tiempo real de la labor, siendo posible por ejemplo comunicar al operario un cambio de planes, o solicitar apoyo para una mayor rapidez en la realización de una tarea.

1.2.-Punto de partida

Se parte de la idea de que los vehículos cuentan con una línea de comunicación CAN, por donde viaja toda la información de los sistemas electrónicos del vehículo (motor, centralita de frenos, confort, etc.). Además se cuenta con un sistema de comunicaciones GPRS de bajo coste, con un sensor GPS y con un microcontrolador Arduino.

También se contará con un servidor web que almacenará los datos y hará la función de intermediario entre el vehículo y la persona que visualiza la información.

Todos estos elementos deberían ser más que suficientes para alcanzar los objetivos que se mencionan en el siguiente punto.

Se cuenta además con los conocimientos obtenidos de la titulación, en concreto, los que se aplicarán a este campo serán:

- Instrumentación electrónica
- Electrónica analógica
- Electrónica digital y microcontroladores
- Informática
- Tecnología electrónica y de control
- Comunicaciones industriales

1.3.-Objetivos

El trabajo consta de varios objetivos bien definidos:

- En primer lugar la lectura y recogida de datos de la línea CAN de un vehículo agrícola. Estos datos serán relevantes para una persona que supervise remotamente los trabajos de campo, por ejemplo a qué velocidad se está trabajando, revoluciones por minuto del motor del vehículo, labor que se está realizando etc.

- Como añadido a la información recogida, se instalará una antena GPS, aunque en la actualidad existen muchos vehículos de este tipo equipados con sistemas GPS de alta precisión, estos tienen un precio muy elevado. Viendo este trabajo como un sistema de gestión de grandes flotas de vehículos, no es económicamente rentable la instalación de estos sistemas de alta precisión para únicamente realizar una geolocalización. Por lo tanto un sistema GPS de bajo coste será más que suficiente para cumplir el objetivo.

- Toda la información recogida de la línea CAN, junto con los datos adquiridos a través del sistema GPS, se enviaran a través de las redes de telefonía móvil a un servidor conectado a Internet. Para ello se empleará un sistema de bajo coste compatible con Arduino. Además este sistema provocará un consumo de datos móviles muy bajo, disminuyendo así el coste para el usuario final.

- La información alojada en el servidor web deberá ser accesible desde cualquier lugar del mundo a través de Internet, de manera que se habrá conseguido conectar el vehículo agrícola con la red.

Con todos estos sistemas se espera obtener una solución funcional y de bajo presupuesto, que sea capaz de mejorar la gestión que las empresas realizan de sus flotas agrícolas, así como optimizar los trabajos de campo, repercutiendo esto directamente en un ahorro en los costes que supone llevar a cabo estas actividades.

1.4.-Metodología de trabajo

Para el alcance de los objetivos inicialmente propuestos, se han realizado una serie de tareas en orden, de manera que se complementen unas con otras. Estas actividades se enumeran y detallan a continuación:

- **Tarea 1:** Estudio de los buses de comunicación empleados en maquinaria agrícola.
  En esta etapa se ha aprendido el funcionamiento de las líneas CAN e ISOBUS, de forma que se han obtenido los conocimientos para posteriormente realizar la comunicación y obtener los datos.

- **Tarea 2:** Montaje de Arduino y módulo de comunicaciones CAN.
  En esta segunda etapa, se ha realizado el montaje del primer sistema que se utilizará para dotar al microcontrolador Arduino de un enlace de comunicaciones CAN. Debido a que la comunicación entre Arduino y el dispositivo de comunicación CAN se realiza mediante un bus SPI, el alumno se ha documentado sobre ello para poder realizar la conexión de manera correcta.

- **Tarea 3:** Conexión del sistema anterior a un vehículo y obtención de datos.
  Se ha realizado la conexión con un tractor New Holland T7 270. Se ha observado el comportamiento de la línea CAN del vehículo. Se ha realizado un programa para filtrar mensajes y poder obtener los datos deseados. Además en esta etapa se ha realizado el programa que decodifica los mensajes en el bus CAN para que puedan ser entendidos por el usuario final.

- **Tarea 4:** Montaje y prueba de sistema GPS.
  Llegados a este punto, se ha añadido al sistema empleado anteriormente un módulo de comunicación GPS. Se ha probado su funcionamiento y se ha aprendido a interpretar la información que el GPS envía al microcontrolador.

- **Tarea 5:** Estudio sobre servidores web.
  Se ha hecho un estudio exhaustivo sobre el funcionamiento de un servidor web. Se ha aprendido a programar en lenguaje PHP, además del funcionamiento de las bases de datos MySQL. Al finalizar esta tarea se tiene un servidor web online y totalmente funcional.

- **Tarea 6:** Pruebas de comunicación GPRS.
  Se ha terminado de montar el prototipo añadiendo el módulo SIM800l, que dota de comunicación GSM/GPRS al sistema. Tras hacer las pruebas pertinentes, se consigue enviar información al servidor web creado en la tarea anterior.

- **Tarea 7:** Programa final.
  Una vez terminado el montaje, se ha realizado el programa final de todo el prototipo, probando su correcto funcionamiento en un vehículo real (New Holland T7 270). Con este apartado quedan cumplidos los objetivos inicialmente planteados.

- **Tarea 8:** Otros.
  Aunque no se han construido por ausencia de necesidad para el proyecto, se ha diseñado una fuente de energía para poder alimentar todo el sistema con la batería de un vehículo, y una caja que podría contener al sistema final tras pasar la fase de prototipo. Se han empleado para ello programas CAD (diseño asistido por ordenador), en concreto los software SolidWorks y KiCAD.

- **Tarea 9:** Memoria
  Finalmente se han redactado la memoria, anexos, pliego de condiciones y presupuesto.

A continuación se presenta un diagrama de GANTT en el que se ve la distribución del tiempo utilizada para la realización del trabajo.

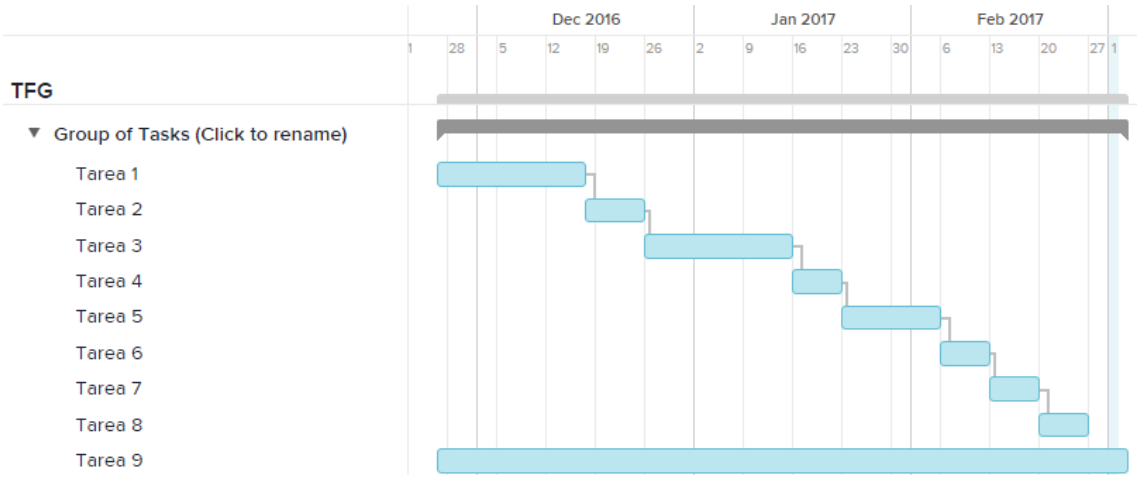


Ilustración 1: Diagrama GANTT

Como se puede comprobar, todas las tareas están correlacionadas, es decir, una no comienza hasta terminar la anterior, salvo en el caso de la memoria que se ha ido redactando conforme se realizaban avances.

1.5.-Glosario de abreviaturas

- **SPI:** Serial Peripheral Interface. Estándar de transmisión de datos entre circuitos integrados en sistemas electrónicos. Trabaja de manera síncrona. Permite el multiplexado de la activación de dispositivos para comunicar con varios dispositivos simultáneamente.

- **GPS:** Global Positioning System. Sistema desarrollado por el departamento de defensa de los EEUU, permite el posicionamiento de un objeto en cualquier punto de la tierra mediante triangulación con ayuda de 24 satélites.

- **NMEA:** National Marine Electronics Association. Organizacion estadounidense que establece los estándares de la electrónica marina. Desarrolló el estándar por el que se comunican los receptores GPS.

- **GPRS:** General Packet Radio Services. Sistema de comunicación dúplex que funciona de manera inalámbrica. Permite realizar comunicaciones por internet, además de otros servicios como mensajes cortos o mensajes multimedia.

- **SIM:** Subscriber Identity Module. Tarjetas utilizadas para la identificación de un usuario en una red de telefonía móvil. Estas tarjetas pueden desmontarse y colocarse en diferentes terminales de manera que un usuario puede acceder a la red de diferentes maneras.

- **CAN:** Controller Area Network. Protocolo de comunicaciones basado en una topología de bus. Muy extendido en la industria del automóvil y la maquinaria agrícola e industrial por su robustez.

- **PHP:** Lenguaje de programación de lado del servidor. El código PHP puede estar embebido en código HTML, potenciando este último.

- **MySQL:** Sistema de gestión de bases de datos open source desarrollado por Oracle Corporation. Es el sistema más popular del mundo en este ámbito.

- **HTML:** HyperText Markup Language. Lenguaje de programación web. Con él se define como se visualizará o comportará una página web.

1.6.-Definiciones

- **Arduino:** Compañía de hardware libre que desarrolla placas basadas en microcontrolador de bajo coste. También produce el entorno de desarrollo para estas placas.

- **Telemática:** Palabra derivada de "telecomunicaciones" e "informática". Es la disciplina que une estos dos campos, implementando servicios y aplicaciones que hacen uso de ellos.

- **Servidor web:** Se puede referir al ordenador que contiene al programa que procesa una aplicación en el lado del servidor. También puede referirse al programa. Realiza comunicaciones uni o bidireccionales de manera síncrona o asíncrona.

- **Implemento agrícola:** Sistema o máquina que se puede añadir a un vehículo agrícola como un tractor para realizar una labor determinada (Arar un campo, abonar, etc.)

- **Conector Deutsch:** Tipo de conector utilizado como estándar para la conexión de implementos a las líneas CAN de un vehículo agrícola. También empleado para diagnosis de errores.

- **Interfaz:** Permite el intercambio de información entre dos sistemas diferentes, o entre un sistema y una persona.

- **Adquisición de datos:** Proceso por el cual se mide un fenómeno eléctrico o físico y a partir de él se obtienen datos experimentales que proporcionan una información veraz.

- **Máquina de estados:** Estructura de programación que definirá el comportamiento de un sistema en función del estado en que este se encuentra. Cada estado define las condiciones de funcionamiento del sistema a partir de una serie de variables.

- **Prototipo:** Modelo o primera aproximación de un invento, a partir del cual se puede crear un producto final.

- **Hardware:** Elementos físicos que constituyen un sistema electrónico o informático.

- **Software:** Conjunto de programas que definen el comportamiento de los elementos físicos de un sistema electrónico o informático.

2.-Hardware

En este apartado se describen los elementos hardware empleados en el desarrollo del prototipo.

2.1.-Arduino UNO

Es una placa de desarrollo basada en el microcontrolador ATmega328P, con 14 entradas y salidas digitales. Cuenta también con 32KB de memoria interna, 2KB de SRAM y 1KB de EEPROM. También incluye un puerto de serie entre los pines digitales 0 y 1, además de un bus SPI en los pines 10, 11, 12 y 13. Se utilizará este bus para las comunicaciones del módulo CAN con Arduino, ya que se pueden conseguir velocidades bastante elevadas por SPI.

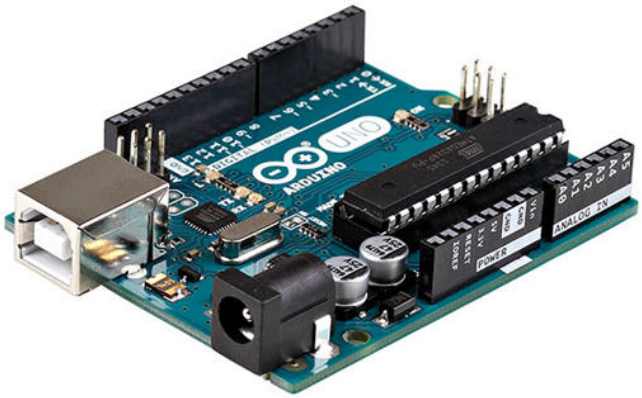


Ilustración 2: Arduino UNO [2]

Se ha decidido utilizar este microcontrolador por su bajo coste, además de la comunidad que le brinda un gran soporte debido a su naturaleza de hardware libre.

2.2.-Interfaz CAN

Debido a que Arduino UNO no cuenta con un bus de comunicaciones CAN, se hace necesario añadir un módulo externo que dote al sistema de esta comunicación.

Para el presente proyecto, se ha elegido un shield con todos los componentes ya montados.

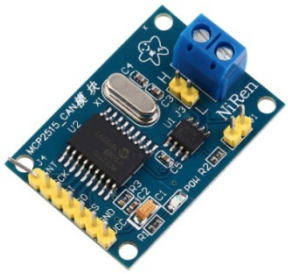


Ilustración 3: Interfaz Arduino - CAN bus

Este shield incluye diferentes componentes:

- MCP2515: Controlador autónomo CAN. Compatible con CAN 2.0A y CAN 2.0B, por lo que puede trabajar con tramas extendidas y con tramas estándar. Tiene una interfaz serie SPI para comunicarse con el microcontrolador.

- MCP2551: Es un transceiver CAN de alta velocidad (hasta 1MB/s) y hará de puente entre el controlador MCP2515 y el propio bus físico CAN. Implementa compatibilidad con el estándar ISO-11898.

- Cristal oscilador de 8 MHz: Es el encargado de sincronizar el controlador con el bus, de manera que los bits se envíen y reciban a la velocidad correspondiente.

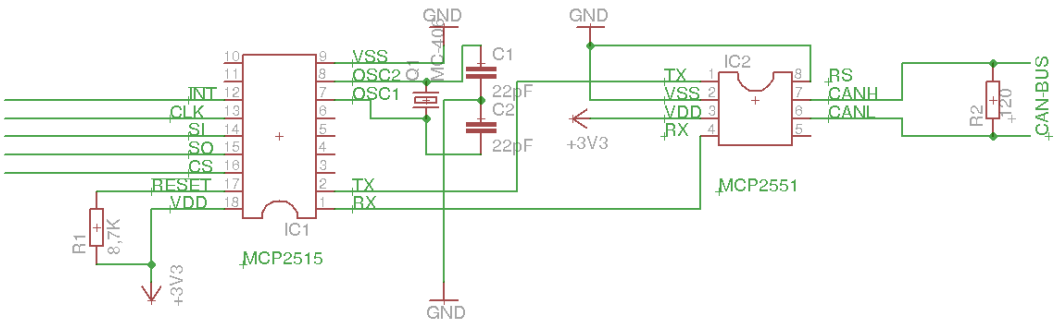


Ilustración 4: Esquema eléctrico del módulo CAN [3]

Este sistema presenta la ventaja de tener una gran sencillez para su conexión con Arduino, además de facilitar el desarrollo del software posterior al ser compatible con librerías de protocolo CAN.

2.3.-Módulo GPS

El controlador Arduino UNO no presenta soporte nativo para conectividad GPS, por lo que es necesario añadir un sistema externo que proporcione esta funcionalidad.

Para la geolocalización del vehículo se ha utilizado un módulo de GPS NEO-6M similar al de la imagen.

Ilustración 5: GPS NEO-6M [4]

Este módulo funciona con una alimentación de 5V, y se comunica con Arduino a través de un puerto de serie. El puerto funciona a una velocidad de 9600bps y transmite la información a través del estándar NMEA. Además este módulo puede medir velocidades de hasta 500m/s.

2.4.-Módulo GSM/GPRS

Gracias a la capacidad de expansión que posee el sistema Arduino UNO, es posible dotarlo de comunicación GSM/GPRS para el envío de los datos recogidos anteriormente a la red. Esto se realizará mediante un módulo con conectividad GSM y GPRS; en este caso, el modelo elegido ha sido el SIM800l.

Este módulo tiene un coste muy bajo y cuenta con conectividad cuatribanda y capacidad de realizar peticiones HTTP, lo cual es más que suficiente para el objetivo que se le va a encomendar.

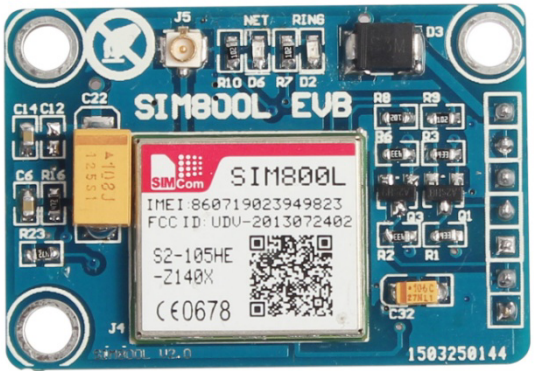


Ilustración 6: Shield SIM800l empleado

Se insertará una tarjeta SIM de un operador cualquiera en la ranura dispuesta para ello. El propio módulo tiene asignado un número IMEI, por lo que podrá iniciar sesión en la red de telefonía de cualquier compañía como si de un teléfono móvil se tratase.

Este sistema presenta una gran ventaja, puesto que la comunicación entre Arduino y el módulo se realizará mediante el puerto serie que lleva integrado el propio módulo. Esto sumado a su reducido precio lo hace perfecto para satisfacer la necesidad planteada. Se utilizarán los comandos llamados “AT HAYES”, explicados en los siguientes apartados del presente documento.

3.-Comunicaciones

En el presente apartado se definen todas las comunicaciones que se utilizarán para el desarrollo del proyecto.

3.1-Bus CAN

El bus CAN (Controller Area Network), es un protocolo de comunicación desarrollado en 1983 por Bosch. Se utiliza en entornos distribuidos para comunicación de distintos nodos en una topología de bus. De esta manera se consigue eliminar cableado, reduciéndolo únicamente a los dos hilos del bus (CAN-H y CAN-L), y a los enganches.

En 1993 se estandarizó el bus CAN en la norma ISO 11898, en ella se recoge el estándar CAN 2.0A, que utiliza identificadores de dispositivo de 11 bits, y el estándar CAN 2.0B, que emplea un formato extendido de identificador de 29 bits. También se recoge el estándar CAN FD, que se está implementando en la actualidad, aunque en el presente proyecto no se va a utilizar.

En la norma también queda definida la máxima velocidad del bus, que será de 1Mbit/s para el bus CAN de alta velocidad, con resistencias finales de 120 Ω; y de 125 Kbit/s para el bus CAN de baja velocidad tolerante a fallos con resistencias de terminación de 100 Ω.

Los niveles lógicos se establecen en un par trenzado de hilos, que reducen el efecto del ruido y las interferencias. Estos dos hilos se denominan *CAN high* y *CAN low*, midiendo la diferencia de tensión entre ambos hilos se determina el estado del bus, este puede ser:

- Estado recesivo: Los dos hilos tienen el mismo nivel de tensión, según la norma ISO 11898, esta tensión estará comprendida entre -2 y 7 V.
- Estado dominante: En este estado existirá una diferencia de tensión en el bus de entre 1.5 y 3 V

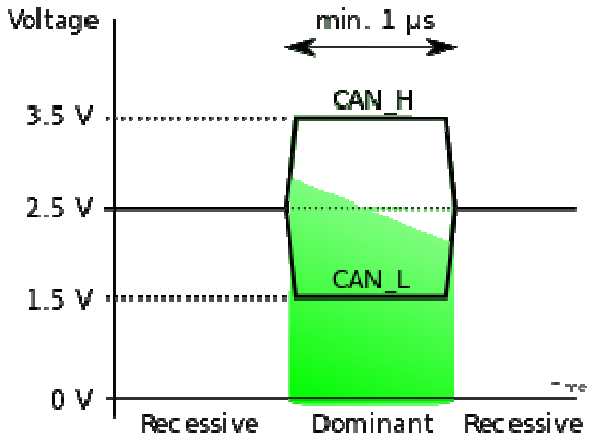


Ilustración 7: Niveles CAN [1]

3.1.1.-Acceso al medio

Como ya se ha mostrado, el bus CAN emplea dos estados para los niveles lógicos 1 y 0, llamando a estos recesivo y dominante, respectivamente.

Es muy importante que todos los nodos de la red estén sincronizados y muestreen los bits al mismo tiempo, puesto que la manera de dar prioridad a uno u otro mensaje es la siguiente:

1. Cuando dos nodos intentan transmitir bits diferentes se produce una colisión. El bus CAN hace que prevalezca el bit dominante, por lo que el nodo que esté transmitiendo el bit recesivo deberá parar la transmisión tan pronto como detecte la colisión.
2. Cuando se transmiten los primeros bits de la trama, es decir el identificador (11 bits para CAN2.0A y 29 bits para CAN2.0B), se produce el arbitraje para acceso al medio. Una vez se haya enviado el identificador solo puede quedar un único nodo transmitiendo. Los nodos que no hayan podido transmitir lo intentarán de nuevo cuando finalice la trama actual.

De todo esto se deduce que los nodos con un valor menor en su identificador tendrán más prioridad que los que tengan valores más altos, puesto que tendrán más ceros en sus primeras posiciones (bits dominantes).

3.1.2.-Sincronización del bus

El bus CAN no tiene una señal de reloj que sincronice a todos los nodos conectados, por lo que aunque todos los nodos funcionen a una misma velocidad, la deriva de reloj, o los retardos introducidos por el propio bus pueden causar problemas.

Por lo tanto la sincronización es crucial, sobretodo en la fase de arbitraje donde todos los nodos transmiten y escuchan al mismo tiempo. Una vez el bus queda libre tras el último mensaje, los nodos empiezan a mandar su bit de inicio y la sincronización se produce en los cambios de bit recesivo a dominante.

Para detectar estos cambios, los nodos dividen cada tiempo de bit en slices y esperan que el siguiente cambio se produzca en una unidad de tiempo que sea múltiplo del tiempo de bit, dependiente de la velocidad del bus. De esta manera, si ocurre una transición fuera del tiempo esperado, se puede ajustar el tiempo para sincronizarse.

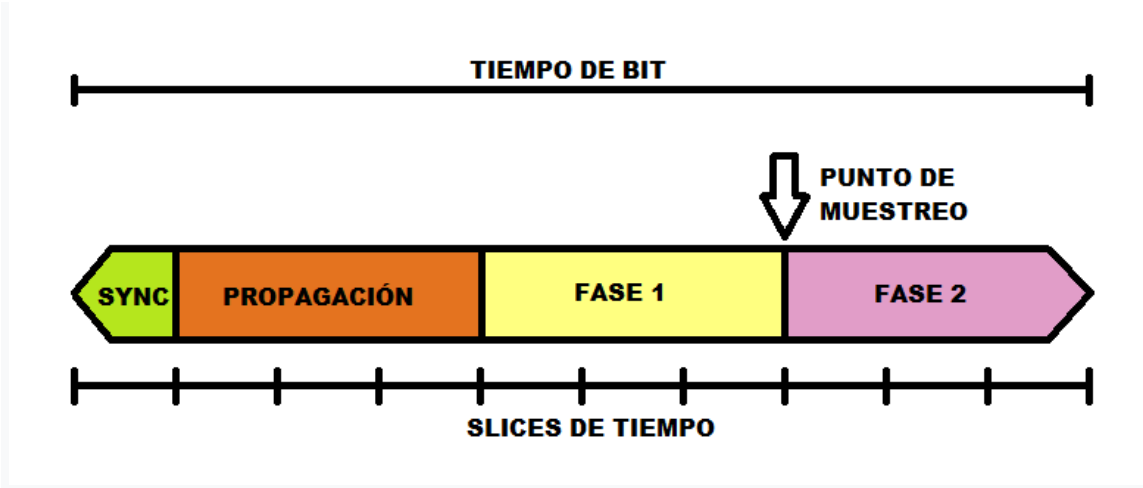


Ilustración 8: Actuaciones en tiempo de bit

En la imagen se ve como se divide un bit en varios slices de tiempo. En esas franjas de tiempo se realizan operaciones para cada bit: sincronismo, propagación, fase 1 y fase 2.

Aproximadamente a un 75% del tiempo de bit se produce la lectura de éste. Así se asegura que el bus esta sincronizado, que los nodos que han perdido el arbitraje han dejado de transmitir, y que el valor leído ya ha sufrido las transiciones necesarias, por lo que debe estar libre de errores.

3.1.3.-Tramas CAN

Como se ha explicado en el apartado anterior, en los buses CAN existen dos tipos de trama, la base, con identificador de 11 bits, y la extendida con identificador de 29 bits. En este caso se va a explicar únicamente la trama extendida, ya que es la que se emplea en el caso de aplicación del presente trabajo.

Una trama CAN con identificador de 29 bits tiene una longitud de 128 bits, a los que hay que añadir los bits de relleno que se añaden cuando se producen 5 bits de un mismo nivel. Estos bits de relleno tendrán el nivel contrario que los 5 anteriores, y su objetivo es garantizar la sincronización entre los módulos. Por lo tanto tratar estas tramas por programa es un proceso bastante laborioso.

Aunque todos los bits de la trama son igual de importantes, puesto que un error en uno de ellos corrompería toda la trama, se mencionarán los más interesantes a efectos de este trabajo.

- El bit 0 marcará el inicio de la trama.
- Los bits del 1 al 12 (11 bits) indicarán la primera parte del identificador, mientras que los bits del 15 al 33 (18 bits) indicarán la segunda parte del identificador
- Los bits del 41 al 105 (64 bits) contienen los 8 bytes de datos y los siguientes 15 bits indican el código de redundancia cíclica (CRC) para comprobar si existen errores en la transmisión.

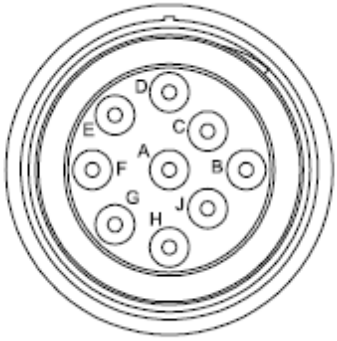
3.2-ISOBUS

En el año 1991 se comienza a trabajar en este protocolo en Estados Unidos y Canadá. ISOBUS nace para universalizar las comunicaciones entre vehículos agrícolas e implementos de distintas marcas, introduciendo una interfaz hombre-máquina que permite al usuario controlar todas las labores desde la misma pantalla, sin necesidad de incluir un nuevo equipo por cada implemento.

Este protocolo está basado en el bus CAN, incluyendo dos líneas de este tipo, una para gestionar el motor y toda la parte mecánica del vehículo, la otra para controlar los implementos que se añaden.

Estas dos líneas CAN están unificadas por una centralita (ECU), por lo que la información podrá pasar de una línea a otra siempre que sea necesario.

La norma ISO11783 define un conector de tipo DEUTSCH para sus conexiones, estableciendo también el pineado de dicho conector como se muestra en la siguiente tabla:



PIN	Función
A	ECU_GND
B	PWR
C	CAN_H bus tractor
D	CAN_L bus tractor
E	No especificado
F	No especificado
G	No especificado
H	CAN_H bus implemento
J	CAN_L bus implemento

Ilustración 9: Conector Deutsch definido en la ISO11783-2

Como puede comprobarse en la imagen, el bus CAN del tractor está entre los pines C y D, mientras que el correspondiente al implemento se encuentra entre los pines H y J.

3.3-Modelo de referencia OSI

Aunque ISOBUS está basada en CAN, tiene ciertas diferencias con esta última. Viendo el modelo de referencia OSI para los protocolos de red con arquitectura de capas, se pueden ver estas diferencias. Aunque ambas normas coinciden en sus capas más bajas (física y enlace de datos), ISOBUS añade capas superiores. Esto no quiere decir que en CAN no existan, sino que simplemente no están definidas.

Ilustración 10: Comparativa CAN - ISOBUS

Ambas normas definen como capa física el par de hilos trenzados con niveles lógicos recesivo y dominante, como ya se ha visto en apartados anteriores. Sin embargo, ISOBUS añade a esta capa física el uso de dos líneas CAN simultáneamente y el conector de tipo DEUTSCH.

Para la capa de enlace de datos, ambas normas definen las tramas normales y las tramas extendidas con identificadores de 11 y 29 bits respectivamente. Además ambas normas especifican que una trama, incluidos los bits de relleno, no excederá los 150 bits.

En la capa de red, ISOBUS define como deben conectarse las distintas centralitas del vehículo tanto al bus del implemento, como al bus del tractor, y de qué manera se comunicarán entre ellas. Por su parte CAN no define nada en este nivel.

La capa de transporte, en la normativa ISOBUS indica cómo deben ser empaquetados y transmitidos los datos, además de cómo debe ser iniciada y cerrada la comunicación entre nodos. La norma CAN no dice nada en este campo.

En la capa de aplicación, ISOBUS incorpora un terminal universal que hará de interfaz entre el usuario y el vehículo. Este terminal consiste en una pantalla sobre la que se mostrará información relevante del implemento conectado sea cual sea la naturaleza de este, ya que toda la información que se transmite entre ellos está definida en la norma.

Como se puede comprobar, a pesar de ser ISOBUS una normativa derivada de CAN, es mucho más completa que esta última, y su principal misión es la de unificar el protocolo entre distintos fabricantes, garantizando así la compatibilidad de implementos de diferentes marcas con un mismo vehículo.

Esto último tiene una importancia vital para este trabajo, puesto que conocida la información que viaja por los buses del vehículo en una marca, podrá extrapolarse el montaje a vehículos de otros fabricantes.

3.4.-Estandar NMEA para GPS

NMEA es el acrónimo de National Marine Electronics Association, esta agrupación fue formada mucho antes de que existieran los GPS para mejorar la comunicación entre aparatos de navegación electrónicos. Actualmente es un estándar para los receptores GPS, que se comunican siguiendo las normas marcadas. Los mensajes se envían codificados de manera que pueden ser interpretados por el microcontrolador que gestiona el módulo GPS:

Por ejemplo el mensaje "$GPRMC" indica la siguiente información:

$GPRMC,155448.00,A,4227.50538,N,00228.85835,W,0.831,084.4,140217,003.1,W*6A

- $GPRMC:
- 155448: Hora exacta en la que se da la información (15:54:48)
- A : Estado (A:Activo, V:Desactivado)
- 4227.50538,N: Latitud 42 grados, 27.50538' Norte
- 00228.85835,W: Longitud 11 grados, 8.85385' Oeste
- 0.831: Velocidad sobre el suelo en nudos
- 084.4: Ángulo de seguimiento en grados
- 140217: Fecha actual (14/02/2017)
- 003.1,W: Variación magnética
- *6ª: Checksum

Existen muchos más mensajes con otro tipo de información, pero en este caso no será necesario utilizarlos, ya que el mensaje "$GPRMC" da información suficiente para el propósito del presente trabajo.

3.5.-Comandos AT HAYES

En 1981 la compañía Hayes Communications desarrolló una serie de comandos que se utilizarían para la configuración de sus módems. Pronto estos comandos se convirtieron en un estándar, llegando hasta el día de hoy.

Estos comandos serán enviados al dispositivo de comunicaciones a través de un puerto de serie, y en función del comando enviado, el dispositivo realizará una acción y devolverá una respuesta que informará sobre la acción realizada.

Existen comandos de todo tipo, para realizar, contestar y colgar llamadas, para enviar SMS, para realizar peticiones WEB, comandos de configuración, etc.

En posteriores capítulos se mostrará el funcionamiento del conjunto Arduino y módulo GSM con estos comandos, por lo que no se profundizará más en este apartado.

4.-Desarrollo del trabajo

A continuación se describen de manera detallada las tareas realizadas para la consecución del objetivo del trabajo.

4.1.-Adquisición de datos

El primer paso es descubrir como viaja la información a través del bus CAN de los vehículos, para ello se ha montado el siguiente circuito:

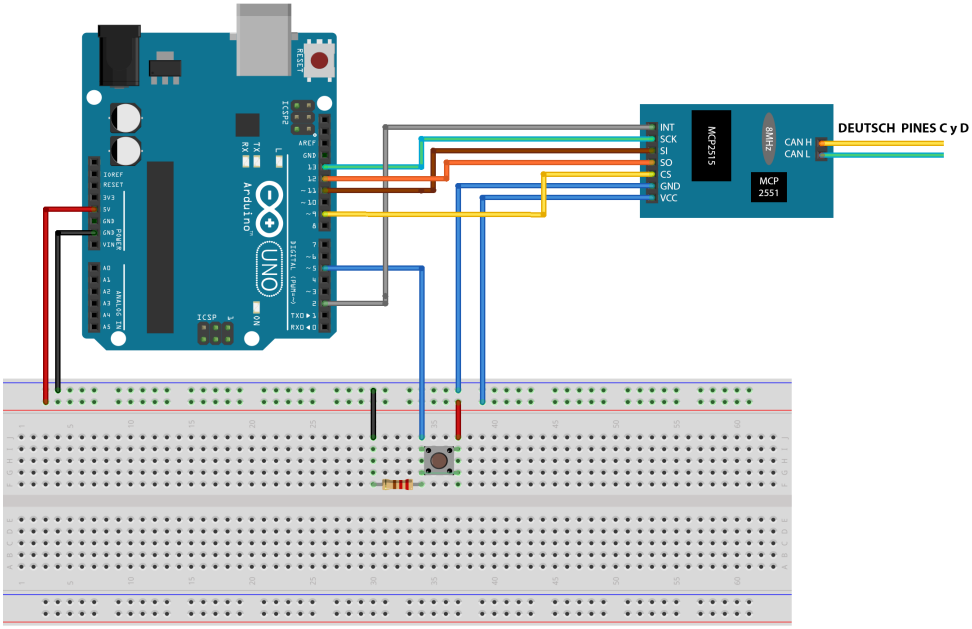


Ilustración 11: Montaje inicial para las primeras pruebas

Este primer montaje consta de una protoboard, el microcontrolador Arduino Uno, el shield CAN bus y un pulsador con una resistencia conectado en configuración pull down. Los dos cables que salen del shield se conectarán directamente a la línea CAN en el conector Deutsch.

Con esta configuración se va a abordar la primera parte del trabajo, la adquisición e interpretación de datos, ya que se desconoce por completo el tipo de datos que viajan por el bus.

Una vez realizado el montaje, con la ayuda de un ordenador portátil, se conectara el sistema al bus y se empezarán a obtener datos.

A continuación se muestran los resultados de las pruebas en diferentes vehículos

4.1.1.-NEW HOLLAND T7 270 BluePower

Para la primera prueba se ha utilizado un tractor similar al que se ve en la siguiente imagen:

Ilustración 12: New Holland T7 [5]

Este vehículo cumple con la normativa ISOBUS, por lo que cuenta con dos líneas CAN, una de gestión de motor, otra para implementos. En este caso únicamente se va a trabajar con la línea CAN del motor.

El tractor cuenta con un conector tipo Deutsch en el interior de la cabina, así como otro de iguales características en la parte trasera. Ambos conectores son válidos para el presente proyecto, puesto que los dos están conectados directamente a las líneas CAN, en el caso del CAN del motor utiliza los pines C y D para CAN alto y CAN bajo respectivamente; en el caso del bus del implemento, se emplean los pines H y J, para CAN alto y bajo respectivamente.

Ilustración 13: Conector Deutsch en la cabina del tractor [6]

En este caso se ha trabajado con el conector interior de la cabina, ya que es más cómodo y accesible que el resto.

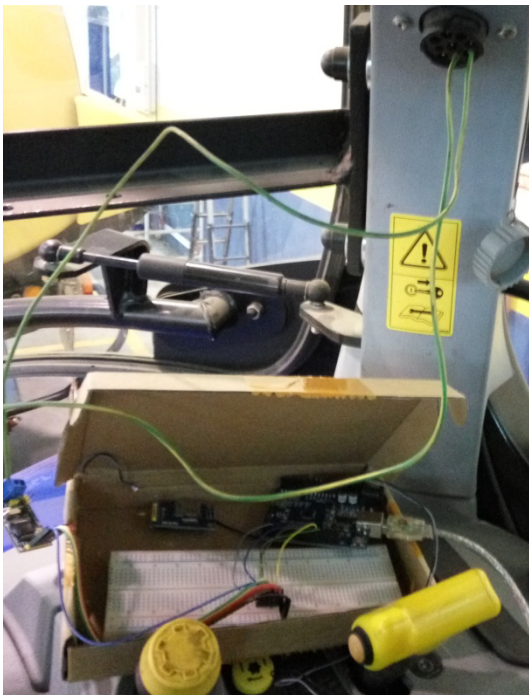


Ilustración 14: Arduino conectado a bus CAN del tractor

Para la realización del software, se ha utilizado la librería "mcp_can.h". Como se ha visto anteriormente, las tramas CAN son muy extensas y sería muy costosa realizar un programa que las transforme en información más sencilla de utilizar. Por suerte esta librería viene configurada por defecto con los pines SPI de Arduino, lo que facilita mucho la tarea puesto que solo habrá que llamar a las funciones definidas dentro de la librería para obtener la información deseada.

Por ejemplo para obtener el identificador CAN que está transmitiendo bastará con llamar a la instrucción "CAN.getCanId();"

Usando esta y otras instrucciones se ha realizado un software muy sencillo que únicamente monitoriza los mensajes de la línea CAN y los muestra en el puerto de serie de Arduino, una vez conectado, se puede ver como todos los mensajes son recibidos a gran velocidad siendo imposible distinguirlos a simple vista, a continuación se muestra una captura de los datos recogidos durante una de las pruebas:

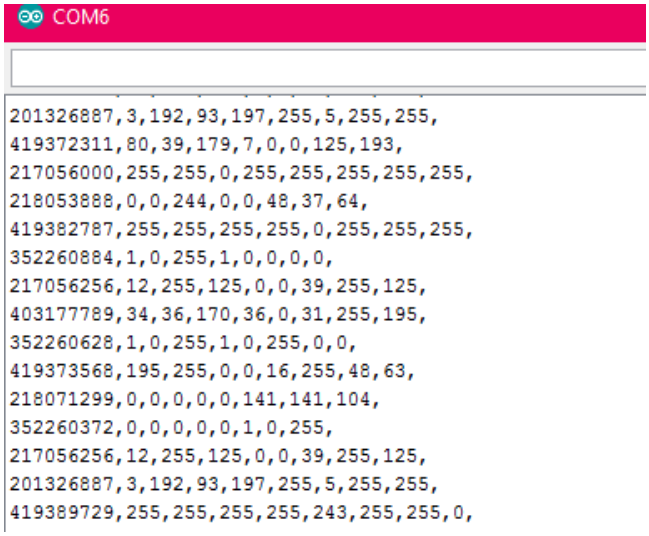


Ilustración 15: Mensajes en el puerto de serie de Arduino

Los mensajes se muestran con el siguiente formato:

ID (29 bits)	B0	B1	B2	B3	B4	B5	B6	B7

Ilustración 16: Formato de los mensajes

En primer lugar aparece el identificador de 29 bits, que se define en la norma CAN2.0B, seguido aparecen separados por comas 8 bytes con la información que se está transmitiendo en formato decimal.

Estos mensajes llegan al puerto de serie a una gran velocidad, por lo que se ha tenido que realizar un análisis de los datos:

- En primer lugar se han recogido durante diez segundos únicamente los identificadores de todos los mensajes que se envían por la línea.
- Después se han introducido todos los identificadores en un programa gratuito de tratamiento de texto, que indica cuantas veces se ha repetido cada identificador.

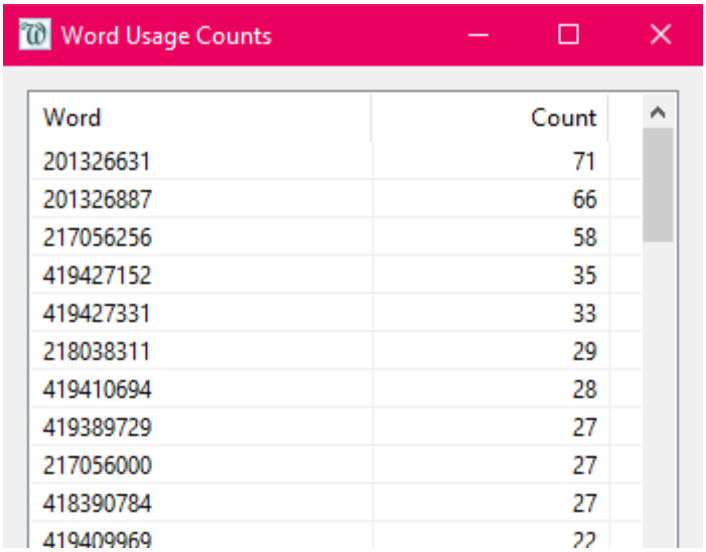


Ilustración 17: Cuenta de uso de palabras con yWriter

- Con este método, se han descubierto unos 50 identificadores diferentes, por lo que el siguiente paso es analizar qué información transmite cada uno de ellos, ya que la información viaja codificada

Una vez obtenidos los identificadores, se ha preparado un programa sencillo para Arduino. Se colocan todos los identificadores en un vector, ordenados de menor a mayor, debido a que se sabe que los identificadores de número más bajo tienen mayor prioridad, por lo que la información que transmiten puede ser más interesante.

Este programa cuenta con un pulsador, que a cada pulsación (gestionada por un sistema antirebote), mostrará el siguiente identificador en el vector, de manera que podrá verse en el puerto de serie toda la información de un mismo identificador, pudiendo observar así las variaciones que ocurran en los mensajes.

La forma de actuar ha sido la siguiente: Se muestra un identificador y se van variando parámetros del tractor (rpm, encender y apagar luces, quitar freno de mano, meter una marcha…), una vez comprobada la información del identificador actual, se toma nota y se pulsa el botón para pasar al siguiente identificador y hacer la misma operación.

Con este método se han podido identificar los siguientes casos:

- **Estado de las luces**
  - En el identificador 419372311, se ha observado que en el cuarto byte, el bit 5 indica el estado de las luces, siendo 0 apagado y 1 encendido

- **Punto muerto**
  - Esta información la da el identificador 217055747, el primer bit del primer byte indica el estado de punto muerto (0), o de marcha (1)

- **Revoluciones por minuto**
  - Este valor ha sido bastante complicado de identificar, puesto que se encuentra codificado en dos bytes, observando el identificador 217056256, se descubre

que los bytes 3 y 4 varían mucho de valor, de modo que se toman valores a diferentes revoluciones de motor para intentar ver la relación.

850 rpm	1000rpm	1230 rpm
217056256,0,255,137,160,26,39,243,137,	217056256,46,255,138,96,31,39,243,138,	217056256,46,255,139,152,38,39,243,139,
217056256,16,255,137,152,26,39,243,137,	217056256,46,255,138,92,31,39,243,138,	217056256,62,255,139,144,38,39,243,139,
217056256,16,255,137,148,26,39,243,137,	217056256,14,255,138,116,31,39,243,138,	217056256,110,255,138,172,38,39,243,139,
217056256,0,255,137,156,26,39,243,137,	217056256,78,255,138,80,31,39,243,138,	217056256,94,255,139,132,38,39,243,140,
217056256,0,255,137,160,26,39,243,137,	217056256,46,255,138,96,31,39,243,138,	217056256,30,255,139,156,38,39,243,139,
217056256,48,255,137,128,26,39,243,137,	217056256,46,255,138,100,31,39,243,138,	217056256,46,255,139,152,38,39,243,139,
217056256,32,255,137,144,26,39,243,137,	217056256,62,255,138,92,31,39,243,138,	217056256,62,255,139,144,38,39,243,139,
217056256,16,255,137,152,26,39,243,137,	217056256,46,255,138,96,31,39,243,138,	217056256,94,255,138,180,38,39,243,139,

Ilustración 18: Comparativa RPM, información del identificador

Tras analizar los datos, se descubre que las rpm vienen dadas por la siguiente fórmula:

$$rpm = \frac{byte4 * 256 + byte3}{8}$$

Se programa esta fórmula en Arduino para comprobar su funcionamiento y como se puede ver en las siguientes imágenes, los datos obtenidos por el programa son similares a los mostrados por el tractor.

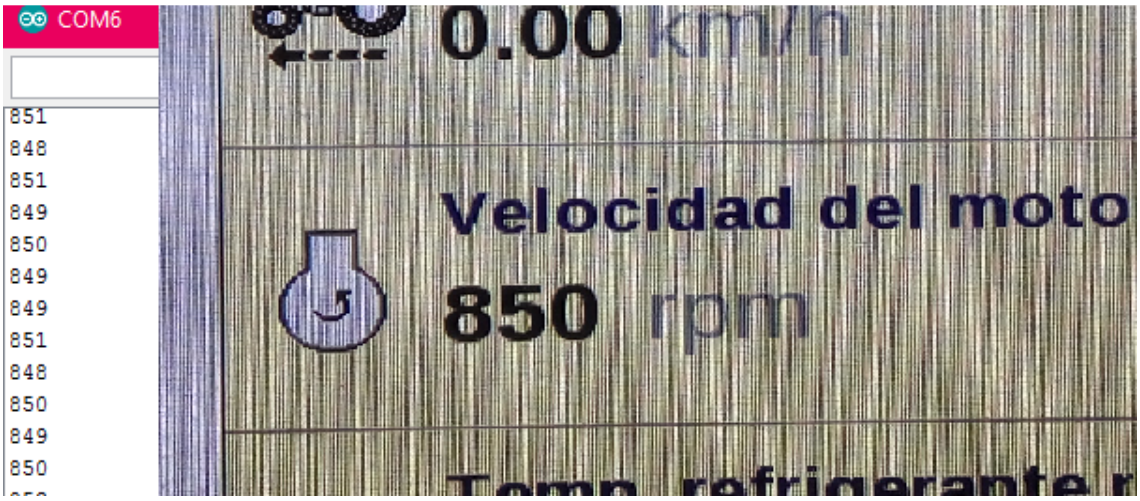


Ilustración 19: Comparativa Valor leído - Valor mostrado

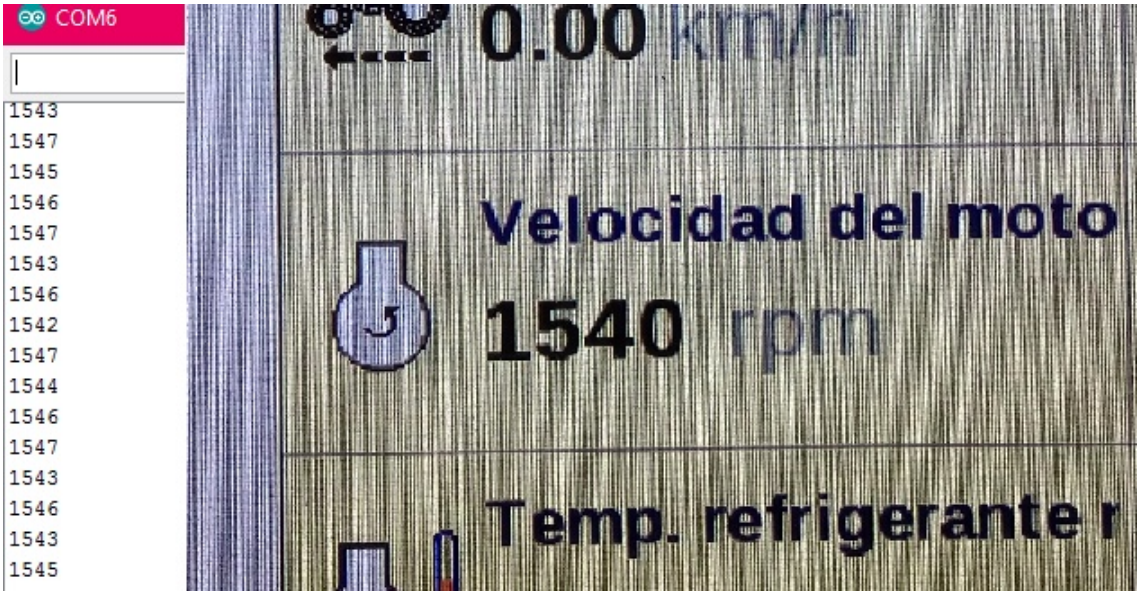


Ilustración 20: Comparativa Valor leído - Valor mostrado

4.1.2.-NEW HOLLAND T5

Tras realizar una segunda prueba en un tractor similar al de la imagen, se comprueba que los valores coinciden y la información obtenida en el vehículo anterior, es perfectamente extrapolable a éste, y posiblemente al resto de vehículos de la marca.

Ilustración 21: New Holland T5 120

4.1.3.-Codificación de información

Para el almacenamiento de la información relevante obtenida a través de la línea CAN, se han creado tres variables (est_marcha, est_luces y est_rpm), que indicarán el estado en tiempo real de la marcha, las luces y las revoluciones por minutos del vehículo.

Todas las variables son de tipo entero, las que informan de un estado tomarán valores 1 o 0 para encendido o apagado respectivamente, mientras que la referente a las revoluciones por minuto, almacenará directamente el valor, tomando los datos de la línea CAN y convirtiéndolos a un formato legible por el usuario empleando la formula definida en el apartado 5.1.1.

Para el resto de información, se deben realizar comprobaciones a nivel de bit, y dado que cada mensaje de can transmite 8 bytes, el primer paso es almacenar el byte que contiene al bit que guarda la información. Este almacenamiento se hace fácilmente gracias a la librearía "MCP_CAN.h", comentada en apartados anteriores. Posteriormente se realizará la comprobación del bit en cuestión, para ello se han empleado máscaras de bit de tipo AND lógico. Por ejemplo si se quiere comprobar si el bit que ocupa la quinta posición en una variable cualquiera, se programarán como condición de un bucle "if" las siguientes líneas en Arduino:

```
if ((marcha & B00000001) != 0){  //Mascara para comprobar si el bit esta activo o no
  est_marcha = 1;
}
else{
  est_marcha = 0;
}
```

Ilustración 22: Máscaras de bit en Arduino

Con este método, es posible comprobar el estado de un bit en concreto, y en función del estado, asignar un valor u otro a una variable.

La información quedaría lista para ser enviada al servidor web donde el usuario podrá visualizarla.

4.2.-Pruebas GPS

Para comprobar el funcionamiento del módulo GPS se ha realizado el montaje del propio módulo con Arduino, con la ayuda de una protoboard como muestra la siguiente ilustración.

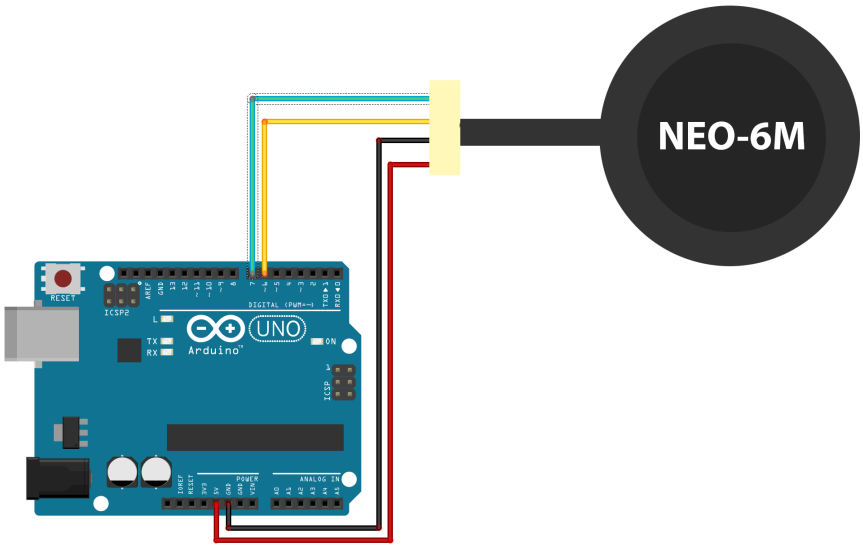


Ilustración 23: Conexionado NEO-6M

Empleando la comunicación NMEA se ha realizado un sencillo programa que recoge la información que el módulo envía por el puerto de serie.

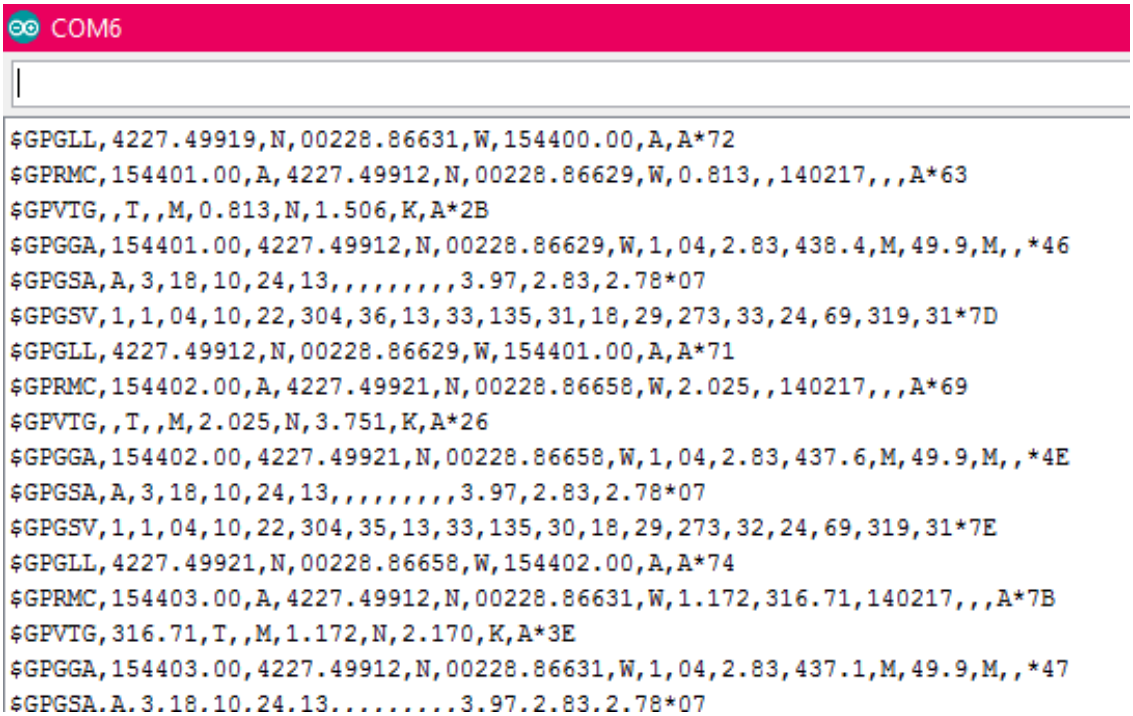


Ilustración 24: Mensajes enviados por el módulo GPS

Se puede comprobar que existen 6 identificadores diferentes en los mensajes, como ya se ha explicado anteriormente, cada identificador de mensaje muestra una información diferente. En el presente caso, el más relevante es el mensaje $GPRMC.

Una vez comprobado el funcionamiento del módulo, se deben seleccionar y tratar los datos para que puedan ser enviados a través de la red GPRS. La realización de un programa que recoja únicamente los datos que queremos sería muy costosa, por suerte, existe una librería OpenSource llamada "TinyGPS" que se encarga de traducir los mensajes en otros más sencillos de utilizar.

4.2.1.-Librería TinyGPS

Esta librería de código abierto facilita el tratamiento de la información suministrada por el GPS. En primer lugar debe definirse un objeto del tipo TinyGPS, que es un tipo propio de la librería con el que después se llamará a las distintas funciones que incluye. En este caso el nombre del objeto TinyGPS será "gps". Se utilizará las siguientes llamadas de la librería:

- Fecha y hora: Se emplea la llamada "gps.get_datetime(&fecha, &hora);", devuelve en las variables de tipo long entre paréntesis la fecha y la hora respectivamente.
- Posición: Se debe llamar a "gps.get_position(&lat, &lon);", devuelve en las variable de tipo long entre paréntesis los valores de latitud y longitud respectivamente.
- Velocidad: con la instrucción "vel = gps.speed();", se guardará el valor de la velocidad actual en centésimas de nudo en la variable llamada vel.

Con estos datos asignados en variables ya se puede trabajar de manera cómoda. El último paso sería convertir la velocidad en nudos a kilómetros hora. Para ello se programa en Arduino la fórmula:

$$\frac{km}{h} = \frac{centésimas\ de\ nudo * 1.852}{100}$$

En la siguiente imagen se muestran estas variables en el puerto serie de Arduino, para comprobar que los datos son totalmente legibles e interpretables.

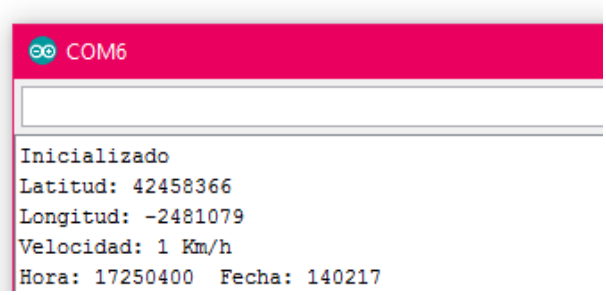


Ilustración 25: Datos mostrados por la librería TinyGPS

Realizados estos pasos, ya se está en disposición de poder enviar por GPRS los datos al servidor WEB.

4.3.-Pruebas GSM

Como el resto de dispositivos utilizados, el módulo SIM800l se alimenta a 5V, la UART que incluye también está adaptada para funcionar con lógica TTL, por lo que en principio no habría problema para la conectividad entre el módulo y Arduino.

Pero existe una particularidad con este módulo, y es que cuando se realizan operaciones de llamada, mensajes, peticiones web, etc. El módulo demanda una corriente de hasta 2A, lo cual no se puede conseguir alimentándolo a través del propio Arduino ni a través del puerto USB de un ordenador. Por ello, es necesario emplear una alimentación externa de 5V y 2A. Es muy importante unir las masas de Arduino y de la fuente, ya que de otra manera la comunicación entre los dispositivos será imposible debido a que tienen que tener la misma referencia.

Se ha definido en Arduino un puerto de serie software, de la misma manera que se hizo con el modulo GPS en el apartado anterior, puesto que el único puerto hardware que incluye Arduino uno se va a emplear para depuración. Este puerto software este definido en los pines digitales 3 y 4; que implementan las conexiones RX y TX respectivamente. A continuación se presenta un esquema de conexionado para las pruebas.

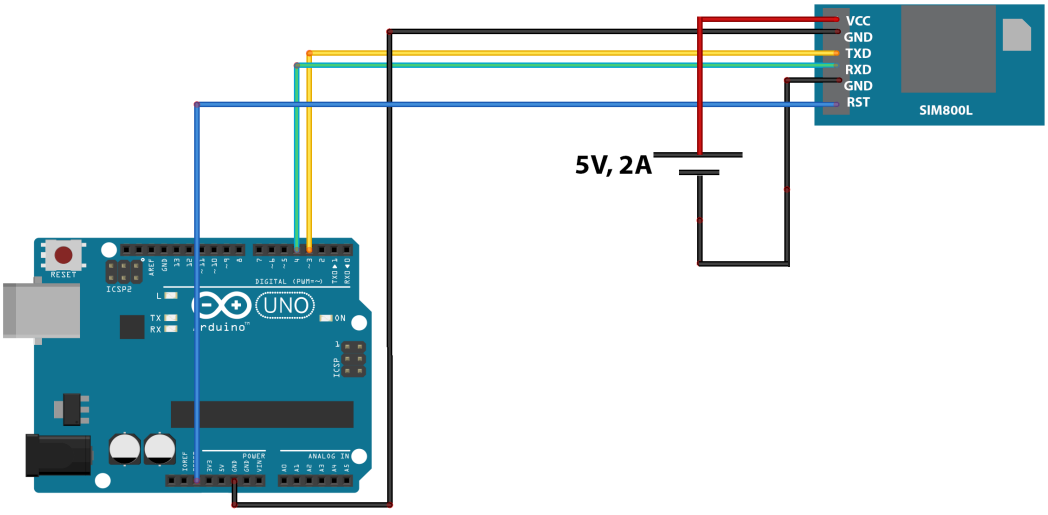


Ilustración 26: Circuito Arduino - SIM800l

Para las pruebas, se ha realizado un programa "puente", esto quiere decir que a través del puerto de serie hardware, se enviarán y recibirán los datos al módulo GSM, de esta manera se aprenderán a utilizar los comandos AT HAYES. Además en el módulo se ha insertado una tarjeta SIM, a la cual se le ha retirado la petición de código PIN, para simplificar los pasos.

El módulo informará de su estado a través de dos leds, uno, siempre fijo indica que la alimentación está conectada, el otro indicará si está conectado o no a la red. Si el led parpadea cada segundo querrá decir que no hay conexión a la red, si parpadea cada 3 segundos, la conexión estará establecida.

En primer lugar se envía el comando más sencillo: “AT”, el módulo responderá únicamente con un “OK”, que indicará que el módulo esta activo y a la espera de instrucciones. Ya que el objetivo de este trabajo es conseguir una comunicación HTTP, se han probado los comandos AT empleados para este fin.

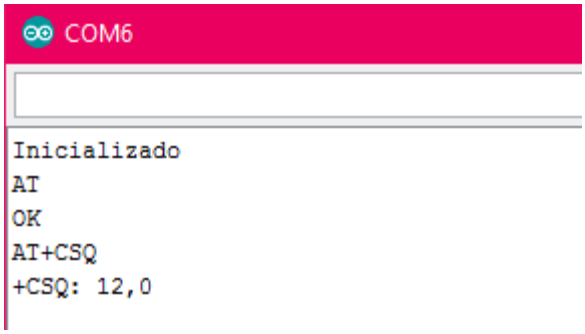


Ilustración 27: Comandos AT HAYES (I)

Con el comando AT+CSQ se comprueba la calidad de la señal, el módulo responde “12”, si se comprueba en la hoja de datos del fabricante, corresponde con una señal de -89dB, que es buena.

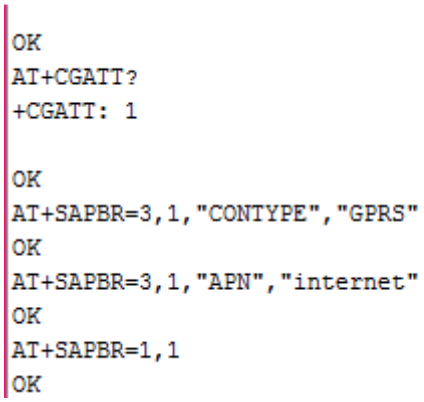


Ilustración 28: Comandos AT HAYES (II)

Con el comando AT+CGATT?, se pide al módulo información sobre la conectividad GPRS, si la respuesta es “1”, querrá decir que hay conexión con la red GPRS.

Los siguientes dos comandos que se ven en la imagen definen el tipo de conexión y punto de acceso que se va a utilizar, y el último valida estos parámetros.

```
AT+HTTPINIT
OK
AT+HTTPPARA="URL","www.google.com"
OK
AT+HTTPACTION=0
OK

+HTTPACTION: 0,302,258
```

Ilustración 29: Comandos AT HAYES (III)

Con HTTPINIT se inicializan los servicios HTTP, el siguiente comando indica que se va a acceder a la URL definida entre comillas. Por último, se envía la petición, la respuesta indica que el comando se ha ejecutado correctamente.

Con los comandos vistos en este apartado es suficiente, puesto que la manera de enviar la información al servidor es mediante peticiones HTTP.

5.-Servidor Web

Dado que se desea centralizar la información recibida de diferentes vehículos agrícolas en un único punto a través de internet, es necesario implementar un servidor web en el que se centralizará la información.

5.1.-WAMP SERVER

Para la realización del servidor web se ha empleado el software para Windows WAMP, que incluye todas las herramientas necesarias (Windows Apache MySQL PHP).
La instalación es muy sencilla, incluye un instalador muy intuitivo en el que únicamente se deben seguir los pasos que se indican en pantalla.

A continuación se detallan estas herramientas y su uso para este trabajo.

5.1.1.-Apache

Es un servidor HTTP de código abierto, multiplataforma y gratuito. Se calcula que aproximadamente el 70% de los servidores del mundo lo utiliza debido a su robustez, seguridad y fiabilidad. Soporta lenguajes de programación como Python, Perl o PHP.

5.1.2.-MySQL

Es un sistema de bases de datos que se distribuye tanto con licencia GPL como con licencia comercial a través de la empresa Oracle. La gran ventaja de esta herramienta radica en que no es necesario que una página web plana tenga toda la información plasmada, ya que puede leerla de la base de datos de una manera dinámica, y mostrar una información diferente en función de las variaciones registradas.

Al igual que Apache, MySQL es compatible con muchos lenguajes de programación desde los cuales se pueden leer y escribir bases de datos, como C++, PERL, Python o más a menudo, PHP, ya que ambos suelen venir ligados.

5.1.3.-PHP

Es un lenguaje de programación de lado del servidor, esto quiere decir que se ejecuta dentro de un servidor cuando un usuario hace una petición, por lo que la respuesta podría ser diferente para distintos usuarios. Su mayor utilización es en el ámbito del desarrollo web, aunque se utiliza también como lenguaje de programación de propósito general.

El código PHP puede estar incrustado en una página web escrita en HTML o en plantillas web. Su parecido con otros lenguajes de alto nivel como C++ lo acerca a personas que conocen estos lenguajes de programación, consiguiendo grandes resultados en poco tiempo.

5.2.-Puesta Online del servidor

Debido a que la mayor parte de equipos informáticos (PCs, routers…) cuentan con cortafuegos y otros sistemas de seguridad, en un principio el servidor web será accesible únicamente desde el ordenador en el que se aloja. A continuación se detallan los pasos seguidos hasta conseguir hacerlo accesible desde internet.

5.2.1.-Red local

Debido a que el servidor web se encuentra dentro de un ordenador privado, en un principio únicamente es accesible desde el propio equipo en el que está alojado, por lo tanto, la única manera de acceder es escribir "localhost" en la barra de direcciones del navegador del equipo.

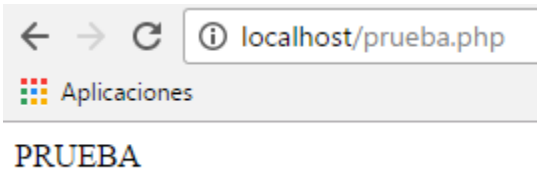


Ilustración 30: Página de prueba en servidor local

Para poder enviar datos desde el vehículo al servidor, es necesario que este último sea accesible desde toda la red.

En primer lugar se debe configurar el servidor Apache para aceptar conexiones externas, para ello, apache cuenta con un archivo de configuración llamado "httpd.conf", se han editado algunos parámetros en este fichero. En la siguiente imagen se ve uno de los parámetros que debe ser modificado. Cambiando la línea que dice "Require local", por "Require all granted", se da permiso a cualquier equipo de la red a acceder al servidor

```
#   onlineoffline tag - don't remove

Require local
```

Ilustración 31: Línea a modificar en "httpd.conf"

Además, se debe permitir el acceso a los archivos del servidor, para ello se buscará en el archivo la sentencia que indica los permisos de los archivos, estará comprendida entre las líneas "<Directory />" y "</Directory>". El texto entre las etiquetas deberá quedar como en la siguiente imagen:

```
<Directory />
Order deny,allow
Allow from all
</Directory>
```

Ilustración 32: Accesibilidad a archivos de Apache

Realizadas estas dos modificaciones, el servidor ya es accesible desde cualquier ordenador de la red local simplemente escribiendo en el navegador la dirección IP local del equipo que almacena la página web.

5.2.2.-Internet

Una vez Apache permite las conexiones desde cualquier equipo, únicamente queda una barrera entre Internet y el servidor local, el router.

El trabajo del router es encaminar hacia cada equipo de la red local los datos correspondientes, para ello emplean los puertos, que están situados en la capa número 4 del modelo de referencia OSI, es decir efectúan el transporte de los datos. Existen 65536 puertos (Del 0 al 65535), algunos de los cuales tienen un fin determinado, por ejemplo el puerto 23 se usa para telnet, o el 80 para HTTP. Este último es el más importante en este proyecto, ya que la manera de comunicar el vehículo con el servidor será a través de peticiones HTTP, luego es necesario que las peticiones HTTP del exterior sean redirigidas por el router al equipo servidor.

En primer lugar es necesario abrir el puerto 80 a la dirección IP local del servidor. En este caso se dispone de un router de la compañía Movistar, al que se accede a través de su IP local, aunque en este caso se ha accedido a la interfaz extendida ya que permite más opciones de configuración. En la barra de direcciones de un navegador se escribe 192.168.1.1:8000, entrando al router a través del puerto 8000 se accede a la configuración extendida.

El siguiente paso es buscar el apartado de configuración de puertos y abrir el número 80 para la IP local del servidor (se puede saber esta IP ejecutando desde cmd en el equipo local el comando “ipconfig –all”), en este caso 192.168.1.37.

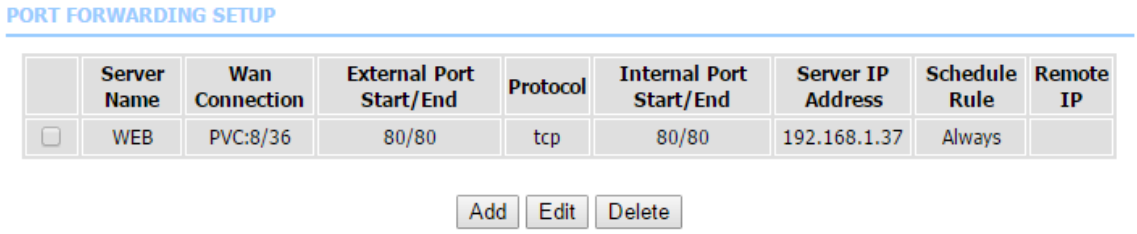


Ilustración 33: Puerto 80 abierto a la IP del servidor

Realizados estos pasos, el servidor debe ser accesible desde la red escribiendo en un navegador la IP pública del router que engloba al equipo servidor.

Ilustración 34: Página de prueba vista desde la red 4G de Yoigo

El siguiente paso es preparar una página WEB capaz de recibir y mostrar los datos cuando se acceda al servidor

5.3.-Programación del portal

Para el funcionamiento del portal se han realizado una serie de programas y acciones que a continuación se detallan.

5.3.1.-Base de datos

Para la construcción del portal WEB, en primer lugar se ha creado una base de datos MySQL que alojará toda la información enviada por el sistema. También desde esta base de datos se leerá la información para mostrarla al usuario cuando acceda al sitio WEB.

Esta tarea se realiza con la ayuda de la interfaz phpMyAdmin, para ello, desde la página bases de datos de la principal de la aplicación, se pulsará sobre el botón "Crear".

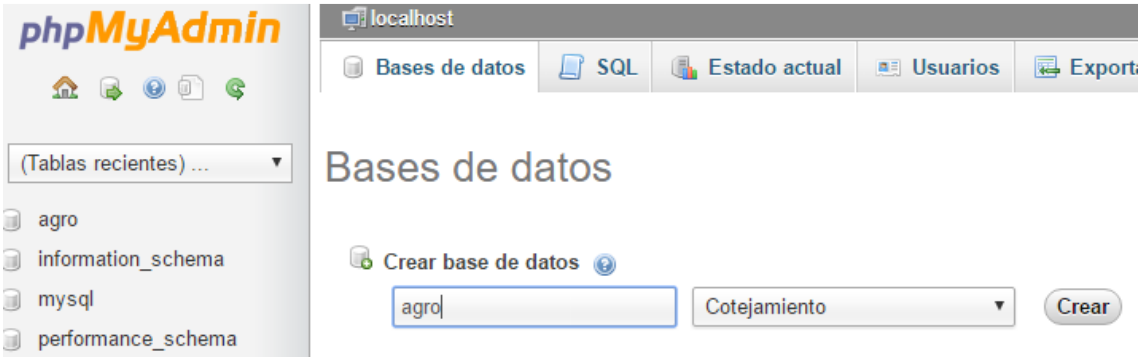


Ilustración 35: Creación de BBDD con phpMyAdmin

Como se puede comprobar, en este caso se ha creado una base de datos con el nombre "agro", dentro de la base de datos se ha creado una tabla con el nombre "parámetros". Esta tabla almacenará los datos en columnas identificándolos por un nombre. En la siguiente ilustración se muestran los nombres para los datos que se guardarán:

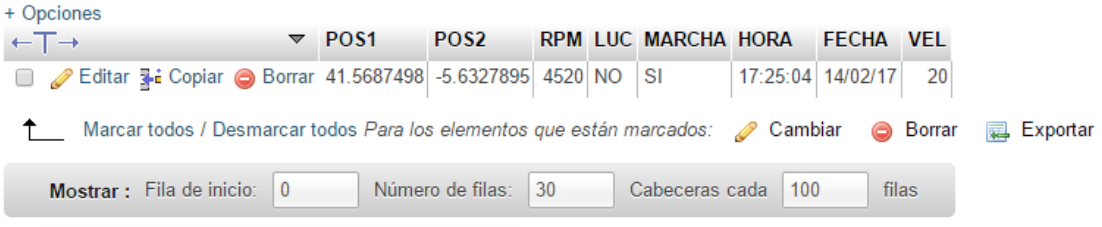


Ilustración 36: Tabla "parametros" en base de datos "agro"

Los campos creados son los siguientes:

- POS1: Dato del tipo "Double". Guarda la información de latitud de la posición del vehículo. Este valor se recoge del módulo GPS.
- POS2: Dato del tipo "Double". Guarda la información de longitud de la posición del vehículo. Este valor se recoge del módulo GPS.
- RPM: Variable de tipo "int". Guarda la información de las revoluciones por minuto del motor del vehículo como un entero. Este valor se recoge de la línea CAN del vehículo.

- LUC: Variable de tipo “char”. Indica si las luces están encendidas o apagadas con la palabra “Si” o “No”. Este valor se recoge de la línea CAN del vehículo.
- MARCHA: Variable de tipo “char”. Indica el vehículo está en marcha o en punto muerto con la palabra “Si” o “No”. Este valor se recoge de la línea CAN del vehículo.
- HORA: Variable de tipo “char”. Indica en una cadena de caracteres la hora a la que se tomo la información que se muestra. Este valor se recoge del módulo GPS.
- FECHA: Variable de tipo “char”. Indica en una cadena de caracteres la fecha en la que se tomo la información que se muestra. Este valor se recoge del módulo GPS.
- VEL: Dato de tipo “int”. Muestra como un entero la velocidad del vehículo en el momento que se enviaron los datos al servidor. Este valor se recoge del módulo GPS.

5.3.2.-Archivos PHP

La interacción entre el usuario y el servidor se dará a través de los archivos PHP. Se han creado dos archivos, uno llamado “datos.php”, que será el encargado de actualizar la base de datos con la información recibida desde Arduino; el otro, llamado “interfaz.php” será el encargado de mostrar la información de la base de datos en pantalla.

Es decir Arduino accederá al archivo “datos.php” y el usuario desde el navegador accederá a “interfaz.php”.

5.3.2.1.-Archivo “datos.php”

El funcionamiento de este archivo es sencillo, el usuario (en este caso Arduino), accederá a una URL, en la que se pueden incluir todos los datos que quiere enviar a la base de datos, este archivo será capaz de recoger los datos de la URL y almacenarlos

Ilustración 37: Aspecto de la URL a la que se accede desde Arduino

En este caso y por ser un trabajo académico, el propio archivo “datos.php” incluye ya las credenciales de usuario para poder modificar la base de datos, por lo que cualquier persona podría modificarla. Por lo tanto este archivo simplemente asigna a unas variables internas los valores que aparecen en la URL, para ello en primer lugar, el programa comprueba si una variable en concreto (pos1 en este caso) se ha introducido en la URL mediante la instrucción “if(isset($_GET['pos1']))”, si esa sentencia es verdadera, hace uso del comando de PHP “GET”, por ejemplo, para asignar a una variable llamada “POS1” el dato que aparece en la URL como “pos1” se debe escribir la sentencia “$POS1 = $_GET['pos1'];”

Una vez se han asignado todos los valores a variables PHP, el siguiente paso es incluirlas en la tabla “parámetros” de la base de datos “agro”

```
<?php echo '<p>Datos salvados</p>'; ?>CRLF
<?php  $conexion = mysql_connect("localhost" , "root" , NULL);CRLF
       mysql_select_db("agro",$conexion);CRLF
       if(isset($_GET['pos1'])) {  //Si existe variable en la URLCRLF
           $POS1 = $_GET['pos1'];CRLF
           $sql = "UPDATE parametros SET POS1=$POS1";CRLF
```

Ilustración 38: Conexión y actualización de base de datos

En primer lugar se abre la conexión con la base de datos a través de la instrucción "mysql_connect", se selecciona la base de datos con "mysql_select_db". Para introducir los nuevos datos, se sobrescriben los datos anteriores con la instrucción "UPDATE table", de este modo únicamente se modifican los datos que hayan sido introducidos en la URL, el resto de datos quedan sin modificar.

Una vez hecho todo esto, la base de datos queda actualizada con la información introducida a través de la URL, por lo que el servidor ya está listo para recibir datos del exterior.

5.3.2.2.-Archivo "interfaz.php"

El usuario solicitará este archivo al navegador para ver la información necesaria. Una vez abierto se puede observar algo similar a la siguiente imagen:

Ilustración 39: Interfaz de usuario

Como se puede comprobar, en la tabla se muestran todos los valores almacenados en la base de datos MySQL. Los valores de latitud y de longitud además se asignan a unas variables PHP llamadas "$lat" y "$lon" que se utilizarán para ubicar el mapa.

Gracias a la API de Google se ha podido integrar un mapa de Google Maps. Para poder utilizar un script de mapa de Google hay que solicitar a través de la página web de Google Developers una clave de API. Una vez obtenida la clave gratuitamente se puede incorporar el código que la propia Google proporciona para integrar el script del mapa en un fichero HTML o PHP.

Además se pueden incorporar marcadores con iconos personalizados para mostrar la información sobre el mapa. En este caso se ha dibujado la silueta de un tractor, por ser más visual el marcador de esta manera.

Dado que el código del script que proporciona Google está escrito en Java ha habido que adaptar las variables "$lat" y "$lon" para poder situar el mapa donde es debido. Para ello se ha utilizado la siguiente línea de código JavaScript y PHP:

```
var sitio = {lat: parseFloat('<?php echo $lati;?>'), lng: parseFloat('<?php echo $lon;?>')};
```

Ilustración 40: Inclusión de variables PHP en JavaScript

Se asigna así a las variables Java "lat" y "lng" el valor de las variables PHP "$lat" y "$lon" como una variable de tipo "Float" (coma flotante). Así el mapa ya puede trabajar con los valores de la base de datos MySQL, y cuando se ejecute en el navegador el código contenido en "interfaz.php", el mapa automáticamente se centrará y colocará un marcador en la última posición conocida del vehículo.

6.-Solución final

En este apartado se describe la solución adoptada después de todas las pruebas anteriores.

6.1.-Conexión Hardware

Finalmente, Arduino tiene que realizar 3 comunicaciones por puerto de serie y una comunicación SPI, por lo que los pines utilizados darán servicio a estas acciones. Además se añade otro pin más para la conexión de un botón auxiliar, que permitirá entrar en un modo de diagnóstico o testeo del sistema.

Como el microcontrolador Arduino Uno únicamente dispone de un puerto de serie hardware, y se va a utilizar para el modo de diagnóstico, se han definido dos puertos de serie Software para el GPS y para la comunicación GSM/GPRS.

La siguiente tabla muestra la relación de pines de Arduino Uno para este trabajo:

Pin Arduino UNO	Función
3	SIM800l RX
4	SIM800l TX
5	Botón auxiliar
6	GPS RX
7	GPS TX
2	INT Módulo CAN
9	CS Módulo CAN
11	(SPI) SI Módulo CAN
12	(SPI) SO Módulo CAN
13	(SPI) SCK Módulo CAN

En el capítulo “Planos” se muestra el esquema de conexión final.

6.1.1.-Alimentación del sistema

Dado que este sistema está hecho para funcionar de manera autónoma en un vehículo en movimiento, debe tenerse en cuenta la conexión de la alimentación. Este tipo de vehículos cuenta con baterías de 12V de corriente continua. El sistema diseñado trabaja con 5V, por lo que es necesario diseñar un sistema para obtener la tensión deseada. Además, es necesario que el sistema sea capaz de proporcionar una corriente de 3 amperios, puesto que ya el módulo GSM puede tener picos de consumo de hasta 2 amperios. De este modo se asegura que todo el sistema va a poder funcionar correctamente.

Se ha elegido el circuito integrado LM2576 en su versión con salida de 5V (existen diferentes variantes). Este circuito permite obtener una salida de 5V y 3A con una entrada entre 7 y 40V. Además necesita de muy pocos componentes adicionales para funcionar.
El esquema de conexión se muestra en el capítulo “PLANOS”.

6.2.-Desarrollo Software

Para el software final se han utilizado todos los programas empleados para pruebas, puesto que de ellos ya se podía sacar toda la información.

Ya que el sistema funcionará de manera automática, es muy importante controlar el tiempo, para ello emplea la función de Arduino “millis()”, que devuelve el tiempo de ejecución sin detener el flujo de programa, por lo que comparando dos medidas de tiempo en momentos diferentes se puede obtener el tiempo transcurrido.

Se ha creado una máquina de estados, que será encargada de controlar las acciones a llevar a cabo en cada momento. En el modo de funcionamiento normal, la máquina de estados realizará diferentes acciones, evolucionando entre ellas en función del tiempo medido. El tiempo total del ciclo se puede cambiar sencillamente modificando una sola variable (“tiempociclo”), que incluirá el tiempo en milisegundos. Todo el desarrollo se ha hecho con un tiempo de 180000 milisegundos, por lo que cada tres minutos se enviará nueva información al servidor. Más adelante se detalla este apartado.

Es muy importante mencionar que se está trabajando con 2 puertos de serie software. Como ya se mencionó anteriormente, Arduino cuenta con un puerto de serie Hardware (asociado a los pines digitales 0 y 1), pero además cuenta con la posibilidad de gestionar un puerto de serie software en dos pines digitales cualquiera gracias a la librería “SoftwareSerial”. En el presente caso, el puerto Hardware se utilizará para “debug” del funcionamiento, de modo que quedará ocupado. Por lo tanto el módulo GPS y el módulo GSM deben controlarse con un puerto software. El problema radica en que no pueden estar abiertas dos conexiones a puertos de serie software simultáneamente. Debido a que el presente sistema trabaja secuencialmente gracias a una máquina de estados, este problema se puede solventar con sencillez, simplemente añadiendo a la entrada de un estado concreto las sentencias para abrir o cerrar un puerto, es decir “Serial.begin(9600p.ej)”, y “Serial.end()”.

Así se consigue que cuando es necesario comunicar con uno de los módulos conectados al puerto de serie software, este puerto se encuentre disponible única y exclusivamente para dicho módulo.

En esta parte del desarrollo, lo más importante ha sido generar las tramas que se enviarán, por ello se ha trabajado con variables de tipo String. Para simplificar el proceso, se ha generado una variable String que ya incluye la primera parte de la URL a la que se debe acceder, es decir, "83.53.13.58./datos.php?", a esta cadena se irán añadiendo posteriormente los datos necesarios para poder enviar al servidor.

```
String urlbase ="\"83.53.13.58./datos.php?";
String bpos1 = "pos1=";      //Tramas base para añadir a la trama principal con la IP
String bpos2 = "pos2=";
```

Ilustración 41: Variables de tipo String para enviar al servidor

También se ha declarado una función que realizará el envío de los datos, dicha función incluye los comandos vistos en el apartado “Pruebas GSM” del presente trabajo. Esta función

"enviadatos" deberá incluir como parámetro una variable de tipo String, que contendrá la trama base mostrada anteriormente más el resto de la trama con los datos a enviar. Los datos se enviarán de uno en uno, por lo que a cada ejecución se realizarán ocho llamadas a la función.

El puerto serie Hardware se utilizará para monitorizar el sistema, ya que se ha programado para que continuamente informe de la acción que se está realizando. Por lo tanto con un ordenador conectado por USB al sistema se podrá ver en tiempo real el funcionamiento del sistema.

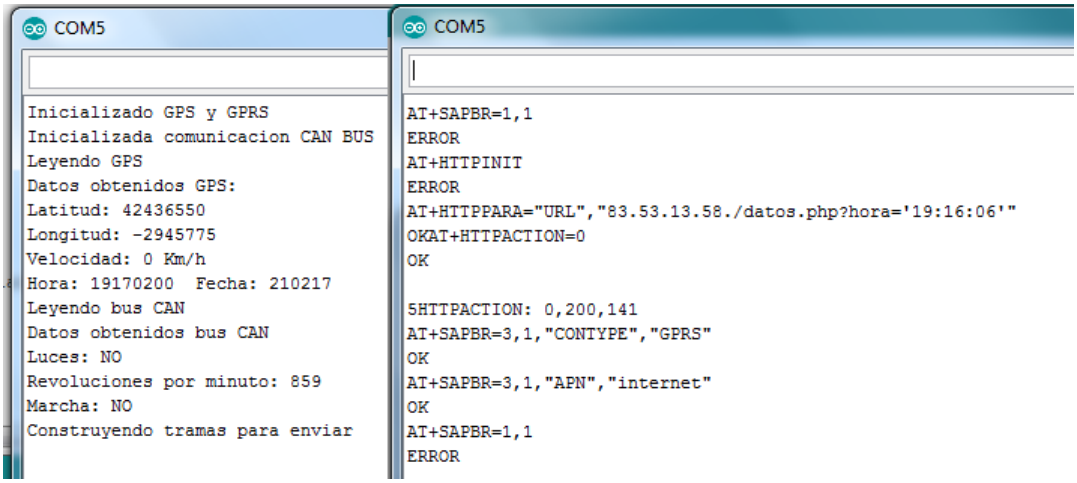


Ilustración 42: Monitorización del sistema

Además se ha añadido la posibilidad de testear el funcionamiento de los diferentes elementos que conforman el sistema. Existe la posibilidad de pulsar el botón auxiliar en cualquier momento de la ejecución del programa y acceder a este modo. Con pulsaciones del botón se irá avanzando por estados de prueba (Prueba de GPS – Prueba de comandos AT – Prueba de línea CAN). Dentro de estos estados, se podrá interactuar con el sistema a través del puerto de serie hardware. El módulo GPS enviará la información bajo el estándar NMEA; por su parte, el módulo GSM permitirá enviar comandos AT y responderá con su estado. Para el caso del módulo de comunicación CAN, se verán los datos de la línea CAN en bruto. Gracias a este modo se podrá identificar el fallo de alguno de los componentes.

```
Prueba modulo SIM8001
Esperando comandos AT
At
OK
AT
OK
AT+HTTP
ERROR
Prueba modulo GPS

$GPGSV,1,1,04,19,,,09,29,,,22,30,,,21,32,,,19*7E
$GPGLL,,,,,,V,N*64
$GPRMC,,V,,,,,,,,,,N*53
$GPVTG,,,,,,,,,N*30
$GPGGA,,,,,,0,00,99.99,,,,,,*48
$GPGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*30
```

Ilustración 43: Modo de prueba de componentes

6.3.-Diagrama de bloques

A continuación se muestra el diagrama de bloques del programa final:

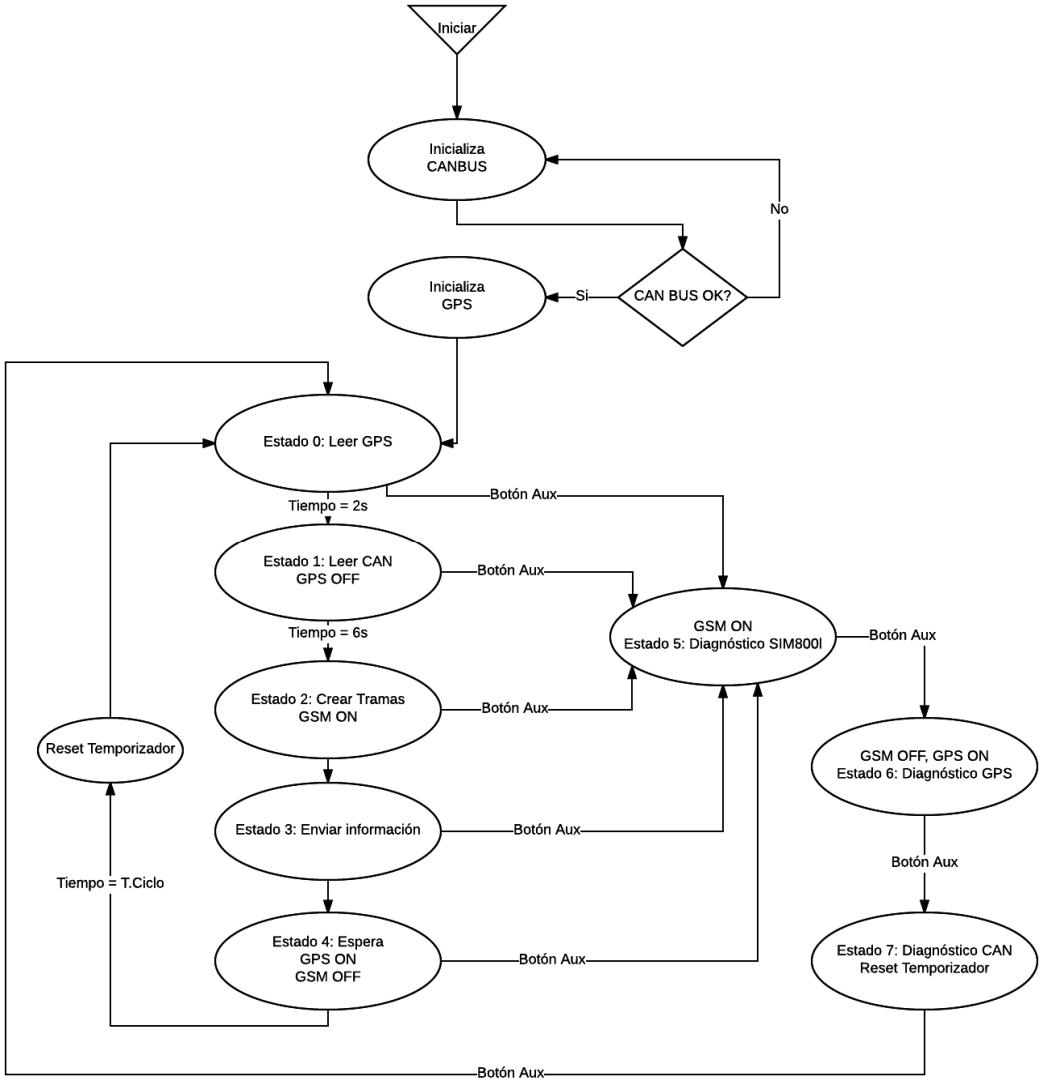


Ilustración 44: Diagrama de bloques empleado en el programa final

Como se puede ver, en un primer momento se inicializa la línea CAN y el módulo GPS, ya que este será utilizado inmediatamente después al entrar en la máquina de estados. La máquina cuenta con 8 estados, cinco de ellos para el funcionamiento normal, los otros tres para pruebas de los módulos.

A continuación se explica la función de los diferentes estados:

- Estado 0: En este estado se lee la información del GPS durante 2 segundos. Antes de entrar se habrán reseteado las variables utilizadas para temporizar, de manera que este estado se toma como inicio de un ciclo completo de trabajo. También antes de entrar se habrá establecido la comunicación con el módulo GPS, y cerrado la comunicación con el módulo GSM.

- Estado 1: Se lee la información de la línea CAN. Al entrar en este estado se muestra en el puerto de serie hardware la información leída en el estado anterior (GPS). También se cierra la conexión con el módulo GPS ya que no se empleará hasta el siguiente ciclo. Una vez pasados 4 segundos dentro de este estado, se habrá leído la información necesaria de la línea CAN.

- Estado 2: El sistema prepara las tramas para añadir a la trama principal y enviar por la red móvil al servidor. Al entrar en este estado se abre la comunicación con el módulo GSM, ya que al pasar al siguiente estado se utilizará. También se muestra la información obtenida en el estado anterior. Se pasará al estado siguiente cuando el sistema haya terminado de preparar las tramas.

- Estado 3: Envío de información. Se muestra en el puerto serie las acciones que se realizan (envío de comandos AT). Se llamará 8 veces (una por cada dato) a la función "enviadato". Se tarda unos 6 segundos en enviar cada dato, por lo tanto en 48 segundos se habrán enviado todos los datos.

- Estado 4: Estado de espera. Sumando los anteriores estados se habrá consumido aproximadamente un minuto de tiempo. El sistema comparará el tiempo que lleva ejecutando el ciclo con el valor introducido en la variable "tiempociclo". Esta variable por tanto deberá ser superior a un minuto para que el sistema tenga tiempo de realizar todas las acciones. Una vez consumido el tiempo se reseteará el temporizador y se volverá al estado 0 para volver a empezar el ciclo. Además se cerrará la comunicación GSM y se abrirá la comunicación GPS.

- Estado 5: Pulsando el botón auxiliar en cualquier estado anterior se accede a este primer estado de diagnóstico para el modulo SIM800l. Al entrar se cerrará la comunicación GPS si está activada y se abrirá la comunicación con el módulo GSM. El sistema esperará en el puerto de serie hardware comandos AT para la prueba del módulo. Pulsando de nuevo el botón se irá al siguiente estado.

- Estado 6: Cuando se alcanza este estado en primer lugar se abre de nuevo la comunicación GPS y se cierra la comunicación GSM. Se realiza aquí la prueba del módulo GPS, que envía al puerto serie toda la información en el formato NMEA. Al pulsar el botón se pasa al estado 7.

- Estado 7: Último estado de diagnóstico. Permite ver los datos de la línea CAN en bruto, tal como se veían en el apartado 5.1.1 antes de codificarlos. Se resetean las variables del temporizador, ya que la siguiente pulsación del botón auxiliar conduce de nuevo al estado 0 para volver a iniciar el ciclo.

6.4.-Prueba de funcionamiento

En la prueba final se ha utilizado un tractor New Holland T6 165, diferente a los utilizados en las primeras pruebas, pero de la misma marca, por lo que el sistema es compatible igualmente.

A continuación se muestran dos capturas de la información mostrada por el vehículo y la información enviada al servidor:

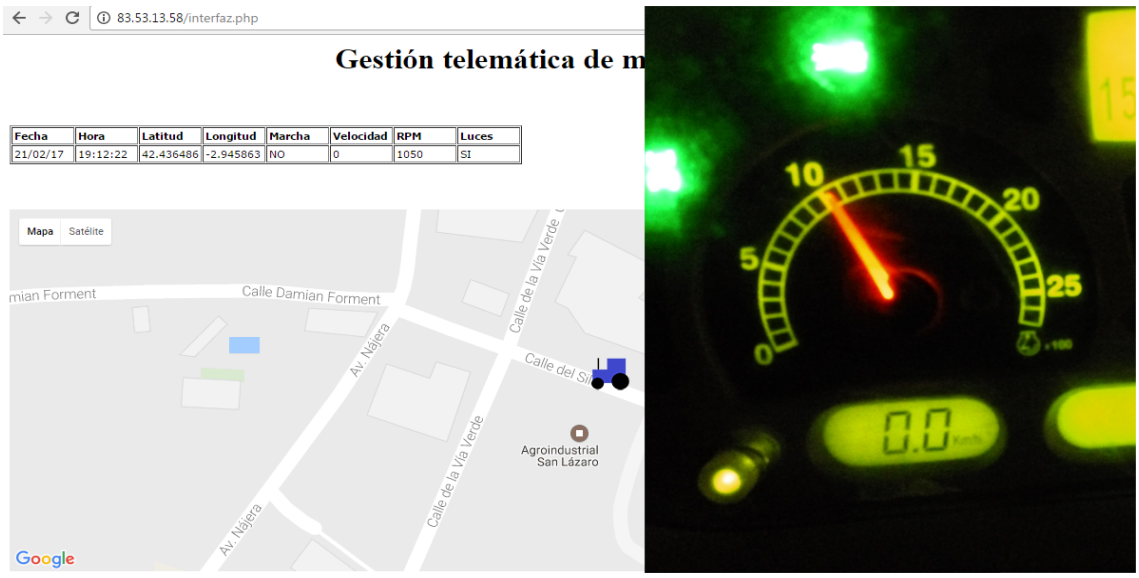


Ilustración 45: Pruebas finales (I)

Ilustración 46: Pruebas finales (II)

Queda comprobado el correcto funcionamiento de todo el sistema, ya que los datos coinciden perfectamente, y el programa no muestra ninguna anomalía al monitorizarlo.

6.5.-Caja para montaje de componentes

Con ayuda del software SolidWorks se ha diseñado una caja para la colocación de todos los componentes del sistema. Esta caja estará compuesta por dos piezas, el cuerpo y la tapa. Contiene agujeros roscados a métrica 3 situados de manera que todos los componentes encajen. Además la caja cuenta con agujeros para la colocación de la antena GSM/GPRS, los cables que se conectarán a la línea CAN del vehículo, y los cables de alimentación junto con el cable de la antena GPS, ya que esta irá situada sobre la tapa de la caja.

La tapa se atornilla al cuerpo de la caja con 4 tornillos de métrica 3, los agujeros del cuerpo ya están roscados para que todo encaje correctamente.

Entre los componentes y el cuerpo se colocarán separadores de métrica 3, para evitar que las conexiones inferiores de las placas toquen con el cuerpo de la caja. Estos separadores tendrán una medida de 5 milímetros.

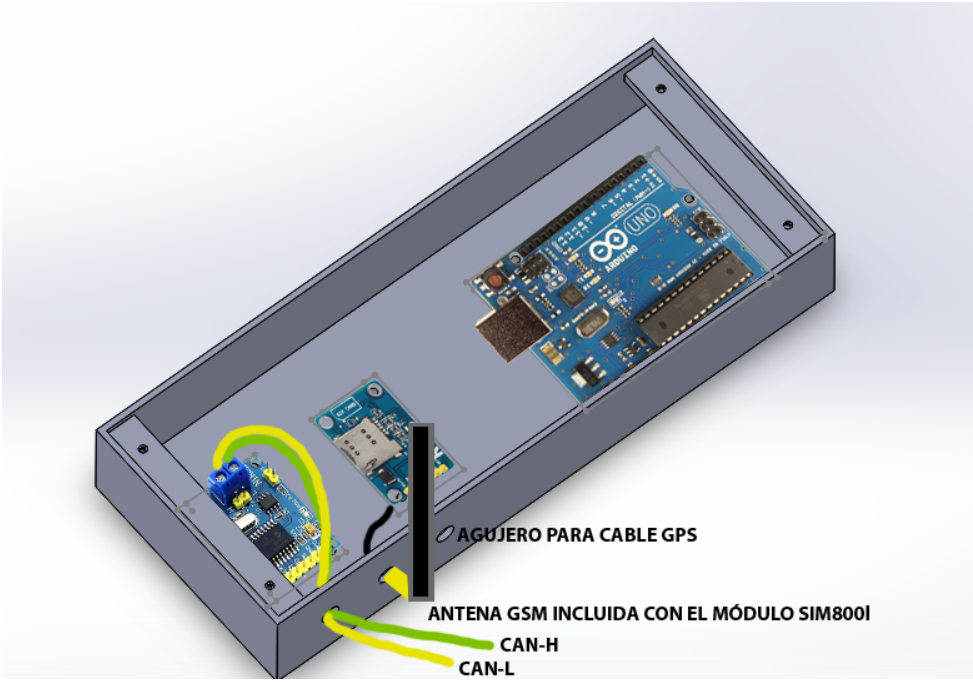


Ilustración 47: Croquis de situación de componentes

Además la tapa contará con una rejilla, de manera que mejorara la circulación de aire por el interior de la caja evitando un calentamiento excesivo de los componentes.

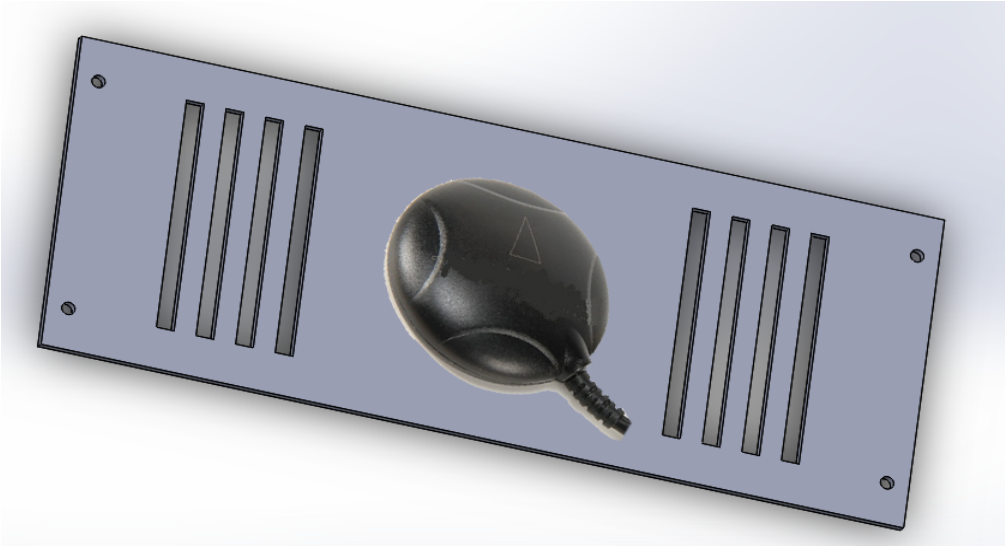


Ilustración 48: Croquis de la tapa con la antena GPS colocada

En el pliego de condiciones se definen los materiales y medios con los que podría ser fabricada esta caja, y en el apartado “Planos” se muestran los planos completos de construcción con medidas.

7.-Conclusiones y puntos de mejora

A la finalización de este proyecto, se ha conseguido cumplir con todos los objetivos propuestos. Se ha obtenido tanto del propio vehículo como de sensores externos información de gran relevancia para el usuario. Asimismo se ha conseguido trasladar esta información en tiempo real a un servidor web, haciéndola accesible desde cualquier parte del mundo.

Como puntos fuertes, cabe decir que se han salvado las barreras de conectividad que pudieran existir por el hecho de encontrarse los vehículos laborando en zonas alejadas de núcleos urbanos. Esto se ha conseguido gracias a la conectividad GSM y GPRS, presente en prácticamente todo el territorio nacional (no así las redes 4G).

Además se ha llegado a una solución felxible y con capacidad de ampliación para atender una gran cantidad de vehículos. La sencillez de programación de las bases de datos MySQL permite añadir nuevos usuarios, asignándoles unas credenciales. Además, el hecho de haber escrito el servidor en PHP permite aumentar fácilmente los datos que se muestran con modificaciones en el software de Arduino muy sencillas. También permite la posibilidad de cambios y adaptaciones en la interfaz (por ejemplo para teléfonos móviles) sin afectar a la programación del sistema montado en el vehículo.

Con esto, se consigue introducir la agricultura en el mundo del internet de las cosas, siendo ahora los vehículos accesibles desde toda la red. Esto supone una ventaja enorme para empresas con grandes flotas de vehículos, puesto que aporta una posibilidad de mejora de la gestión de sus vehículos, ya que por ejemplo, sabiendo donde se encuentran se pueden comunicar con el conductor para que vaya a trabajar a otra finca cercana por ejemplo, o incluso para comprobar la eficiencia de sus trabajadores.

Puesto que en el momento de la realización del proyecto, solo se disponía de vehículos de la marca “New Holland”, sólo ha sido posible probar el sistema en estos. Se desconoce si el sistema funcionaría sin modificaciones en vehículos de otro fabricante, ya que puede que los identificadores de la línea CAN y los mensajes no coincidan. Por esto, hubiera supuesto una ayuda enorme obtener la información referente a la línea CAN del propio fabricante, haciendo muchísimo más sencilla la labor de identificar los mensajes, así como permitiendo ampliar enormemente la cantidad de datos que se envían al servidor.

La forma en la que se ha actuado para identificación de mensajes en la línea CAN ha resultado bastante tediosa, ya que se ha trabajado con un método basado en la experimentación. Dichas pruebas aplicadas a una línea que manda 50 identificadores CAN diferentes resultan en un trabajo muy costoso, y aunque el objetivo era demostrar que es posible realizar comunicaciones CAN con hardware de bajo coste, el resultado habría sido mucho más fructífero si se hubiera contado con la información del fabricante.

Finalmente se ha llegado a una solución totalmente funcional y de bajo coste en cuanto al hardware y tecnología utilizados. Además se trata de un proyecto bastante completo, puesto que incluye diferentes actuaciones de distintos campos (Diseño eléctrico, diseño mecánico, programación de un servidor informático, etc.), por lo que se han complementado los conocimientos adquiridos durante el grado, con conocimientos que perfectamente podrían

cursarse en otros tipos de estudios pero totalmente relacionados, demostrándose una vez más, que el mundo cada día está más conectado gracias a los avances que se realizan en el campo de la electrónica y las telecomunicaciones.

8.-Bibliografía y referencias

[1] https://es.wikipedia.org/wiki/Bus_CAN#/media/File:Canbus_levels.svg

[2] http://www.arduino.org/media/k2/galleries/90/A000066-Arduino-Uno-TH-2tri.jpg

[3] http://www.mikrocontroller.net/attachment/162425/Bildschirmfoto-2.png

[4] http://www.modeltronic.es/images/51452ww.jpg

[5]https://bmcontent.affino.com/AcuCustom/Sitename/DAM/056/T7_225_BluePower_Autocommand_Tier4B_15_040_1.jpg

[6] http://www.pfc-eu.com/uploads/kcfinder/image/ISOBUS%20PUG%20low%20res.jpg

Datasheet MCP2515

Datasheet MCP 2551

Norma ISO 11789 (ISOBUS)

Norma ISO 11898 (CANBUS)

http://www.ti.com/lit/ds/symlink/lm2576.pdf

http://ww1.microchip.com/downloads/en/DeviceDoc/21801d.pdf

http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf

https://es.wikipedia.org/wiki/Bus_CAN

https://de.wikipedia.org/wiki/ISOBUS

https://www.aef-isobus-database.org/isobusdb/login.jsf

http://es.farnell.com/microchip/mcp2515-i-so/can-controller-spi-1mbps-soic18/dp/1292239

https://www.arduino.cc/en/Main/ArduinoBoardUno

https://en.wikipedia.org/wiki/CAN_bus#Data_transmission

https://es.wikipedia.org/wiki/Servidor_HTTP_Apache

https://es.wikipedia.org/wiki/MySQL

https://es.wikipedia.org/wiki/PHP

http://gpsworld.com/what-exactly-is-gps-nmea-data/

http://www.danipasadas.com/php/wampserver.php

http://arduiniana.org/libraries/tinygps/

https://en.wikipedia.org/wiki/PHP

https://geekytheory.com/google-maps-api-v3-placemarks

https://developers.google.com/maps/documentation/javascript/adding-a-google-map

https://es.wikipedia.org/wiki/Conjunto_de_comandos_Hayes

https://cdn-shop.adafruit.com/datasheets/sim800_series_ip_application_note_v1.00.pdf

http://forum.arduino.cc/index.php?topic=116867.0

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Gestión telemática de vehículos agrícolas

DOCUMENTO 3: ANEXO I PROGRAMAS

Alberto Úbeda Cañas

Marzo de 2017

ANEXO I: PROGRAMAS

1.-Prueba Arduino CAN

A continuación se muestra el código utilizado para la separación de identificadores y la identificación de mensajes.

```
#include "mcp_can.h"        //Libreria CAN
#include <SPI.h>
#define ent 5
INT32U canId = 0x000;
unsigned char len = 0;
unsigned char buf[8];
char str[20];
INT32U ids[50] = {150962215, 201326631, 201326887, 201330471,
217055747, 217056000, 217056256, 218038311, 218053888,
218071299, 352260372, 352260628, 352260884, 403112509,
403177789, 418119424, 418119439, 418184960, 418184975,
418381839, 418383107, 419316487, 419316756, 419317027,
419317268, 419321344, 419347712, 419351101, 419356416,
419357696, 419357699, 419360256, 419360512, 419361063,
419361280, 419362048, 419362304, 419362560, 419362819,
419370243, 419370260, 419371523, 419371540, 419372311,
419373302, 419373312, 419373568, 419377175, 419382787,
419389729};
const int SPI_CS_PIN = 9;
int cuentae = 0;    //Cuenta antirebote botón
int enter = 0;
int indice = 0;
MCP_CAN CAN(SPI_CS_PIN); //El módulo CAN comunica por SPI
void setup()
{
Serial.begin(115200);
START_INIT:
if(CAN_OK == CAN.begin(CAN_500KBPS))    //El bus CAN del tractor
trabaja a 500Kbps
    {
        Serial.println("CAN BUS inicializado!");    //Se informa
del estado de la inicialización de la comunicación
    }
    else
    {
        Serial.println("Error CAN BUS");
        delay(100);
        goto START_INIT;          //Volver al intentar
    }
}
void loop(){
    if(CAN_MSGAVAIL == CAN.checkReceive())
    {
        CAN.readMsgBuf(&len, buf);
        canId = CAN.getCanId();       //Se obtiene el
identificador

        if (canId == ids[indice]){    //Si el identificador
corresponde con el ID que se monitoriza, se muestra en el puerto
serie
```

```
            Serial.print(canId);
            Serial.print(",");
            for(int i = 0; i<len; i++){
               Serial.print(buf[i]);Serial.print(",");
            }
            Serial.println();
        }
    }


  if (digitalRead(ent) == HIGH){    //Antirebote boton
    if (cuentae < 10){
        cuentae = cuentae + 1;
        enter = 0;
      }
    else{
        if (cuentae == 21){         //Evita que mientras sigamos
pulsando siga avanzando estados
          cuentae = 21;
          enter = 0;
        }
        else{
          enter = 1;                //El antirebote solo genera
un pulso y se queda esperando a soltar el botón
          cuentae = 21;
        }
      }
    }
  else{
      cuentae = 0;
      enter = 0;
    }

  if (enter == 1){           //Cada pulso de "enter" avanza un
estado
    if (indice < 50){        //Hay 50 identificadores diferentes
      indice = indice + 1;
    }
    else{
      indice = 0;
    }
  }
}
```

2.-Cálculo de RPM con Arduino

Fórmula utilizada para calcular las RPM del motor a partir de los datos del bus CAN.

```
if (canId == 217056256){
  for(int i = 0; i<len; i++){
        if (i == 3){
              bajo = int(buf[i]);  //Guardo byte 3
  }
        if (i == 4){
              alto = int(buf[i]);  //Guardo byte 4
  }
 }
rpm = (bajo+(alto*256))/8;
  Serial.print(rpm);
  Serial.println();
}
```

3.-Codificación de información CAN

Código empleado para codificar la información de la línea CAN de manera que sea legible e interpretable por el usuario.

```
#include "mcp_can.h"
#include <SPI.h>
INT32U canId = 0x000;
unsigned char len = 0;
unsigned char buf[8];
char str[20];
int alto = 0;
int bajo = 0;
int luces = 0;
int marcha = 0;

int est_rpm = 0;
int est_marcha = 0;
int est_luces = 0;
int est_latitud = 0;
int est_longitud = 0;
int est_vel = 0;
int est_fecha = 0;
int est_hora = 0;

const int SPI_CS_PIN = 9;
MCP_CAN CAN(SPI_CS_PIN);
void setup()
{
Serial.begin(115200);
START_INIT:
if(CAN_OK == CAN.begin(CAN_500KBPS))
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}
void loop(){
    if(CAN_MSGAVAIL == CAN.checkReceive())
    {
        CAN.readMsgBuf(&len, buf);
        canId = CAN.getCanId();

        if (canId == 217056256){
          for(int i = 0; i<len; i++){
            if (i == 3){
              bajo = int(buf[i]); //Guardo byte 3
            }
            if (i == 4){
```

```
              alto = int(buf[i]);   //Guardo byte 4
            }
          }
        }

        if (canId == 419372311){
          for(int i = 0; i<len; i++){
            if (i == 3){
              luces = int(buf[i]); //Guardo el byte 5 que
contiene la informacion de luces
            }
          }
        }

        if (canId == 217055747){
          for(int i = 0; i<len; i++){
            if (i == 0){
              marcha = int(buf[i]); //Guardo el byte 0 que
contiene la informacion de marcha/parado
            }
          }
        }
    }
    est_rpm = (bajo+(alto*256))/8;

    if ((marcha & B00000001) != 0){  //Mascara para comprobar si
el bit esta activo o no
      est_marcha = 1;
    }
    else{
      est_marcha = 0;
    }

    if ((luces & B00010000) != 0){  //Mascara para comprobar si
el bit esta activo o no
      est_luces = 1;
    }
    else{
      est_luces = 0;
    }
    Serial.println(est_luces);
}
```

4.-Código Arduino para prueba de módulo GPS.

Primer código empleado con el módulo GPS. Muestra los datos bajo el estándar NMEA.

```
#include <SoftwareSerial.h>

//Creo un puerto de serie Software ya que el hardware lo tengo
ocupado
SoftwareSerial gps(6, 7);

void setup() {
  //Se inicializa la comunicación del puerto de serie de Arduino
para mostrar los mensajes
  Serial.begin(9600);
  while(!Serial);

  gps.begin(9600);//El modulo GPS tiene una velocidad de 9600bps
  delay(1000);

  Serial.println("Inicializado");
}

void loop() {
  if(gps.available()){    //Si el GPS en el puerto de serie
software esta disponible, mostrar en el puerto de serie de
arduino lo que envíe el GPS
    Serial.write(gps.read());
  }
}
```

5.-Código Arduino GPS con datos identificados

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>
TinyGPS gps;    //Defino objeto de tipo TiniGPS
//Creo un puerto de serie Software ya que el hardware lo tengo
ocupado
SoftwareSerial neo6m(6, 7);
long lat, lon;
unsigned long fix_age, time, date, speed, course;
unsigned long chars;
unsigned short sentences, failed_checksum;
void setup() {
  //Se inicializa la comunicacion del puerto de serie de Arduino
para mostrar los mensajes
  Serial.begin(9600);
  while(!Serial);

  neo6m.begin(9600);//El modulo GPS tiene una velocidad de
9600bps
  delay(1000);

  Serial.println("Inicializado");
}

void loop()
{
  while (neo6m.available()) //Si esta disponible el módulo
  {
    int c = neo6m.read();
    if (gps.encode(c))
    {
      gps.get_position(&lat, &lon, &fix_age); //Obtengo latitud
y longitud en centesimas de grado
      gps.get_datetime(&date, &time, &fix_age); //Obtengo fecha
y hora
      speed = gps.speed()*1.852/100;    //Obtengo velocidad

      delay(100);   //Tras un retardo, muestro los datos en el
puerto serie
      Serial.print("Latitud: ");
      Serial.println(lat);
      Serial.print("Longitud: ");
      Serial.println(lon);
      Serial.print("Velocidad: ");
      Serial.print(speed);
      Serial.println(" Km/h");
      Serial.print("Hora: ");
      Serial.print(time);
      Serial.print("  Fecha: ");
      Serial.println(date);
    }
  }
}
```

6.-Código Arduino para prueba de módulo GSM

```
#include <SoftwareSerial.h>

//Creo un puerto de serie Software ya que el hardware lo tengo
ocupado
SoftwareSerial sim800l(3,4);

void setup() {
  //Se inicializa la comunicacion del puerto de serie de Arduino
para mostrar los mensajes
  Serial.begin(115200);
  while(!Serial);

  sim800l.begin(113000);//El modulo sim800l tiene una velocidad
de 9600bps
  delay(1000);

  Serial.println("Inicializado");
}

void loop() {
  if(sim800l.available()){    //Si el sim800l en el puerto de
serie software esta disponible, mostrar en el puerto de serie de
arduino lo que envíe el sim800l
    Serial.write(sim800l.read());
  }
  if(Serial.available()){    //Enviar al sim800l lo que se lea
del puerto serie hardware
    sim800l.write(Serial.read());
  }
}
```

7.-Código PHP "datos.php"

```
<html>
 <head>
  <title>Gesti&oacute;n telem&aacute;tica de maquinaria</title>
 </head>
 <body>
 <?php echo '<p>Datos salvados</p>'; ?>
 <?php  $conexion = mysql_connect("localhost" , "root" , NULL);
                    mysql_select_db("agro",$conexion);
                    if(isset($_GET['pos1'])) {          //Si existe variable en la URL
                              $POS1 = $_GET['pos1'];
                              $sql = "UPDATE parametros SET POS1=$POS1";
                    }
                    if(isset($_GET['pos2'])) {
                              $POS2 = $_GET['pos2'];
                              $sql = "UPDATE parametros SET POS2=$POS2";
                    }
                    if(isset($_GET['rpm'])) {
                              $RPM = $_GET['rpm'];
                              $sql = "UPDATE parametros SET RPM=$RPM";
                    }
                    if(isset($_GET['luc'])) {
                              $LUC = $_GET['luc'];
                              $sql = "UPDATE parametros SET LUC=$LUC";
                    }
                    if(isset($_GET['marcha'])) {
                              $MARCHA = $_GET['marcha'];
```

```
                $sql = "UPDATE parametros SET MARCHA=$MARCHA";
        }
        if(isset($_GET['hora'])) {
                $HORA = $_GET['hora'];
                $sql = "UPDATE parametros SET HORA=$HORA";
        }
        if(isset($_GET['fecha'])) {
                $FECHA = $_GET['fecha'];
                $sql = "UPDATE parametros SET FECHA=$FECHA";
        }
        if(isset($_GET['vel'])) {
                $VEL = $_GET['vel'];
                $sql = "UPDATE parametros SET VEL=$VEL";
        }

        mysql_query($sql);?>
 </body>
</html>
```

8.-Código PHP “interfaz.php”

```
<html>
 <head>
  <title>Gesti&oacute;n telem&aacute;tica de maquinaria</title>
   <style>
     #map {
      height: 400px;
      width: 100%;
     }
   </style>
 </head>
<h1 style="text-align: center;"><strong>Gesti&oacute;n telem&aacute;tica de maquinaria agr&iacute;cola</strong></h1>
<p> </p>
<table border="1" cellspacing=1 cellpadding=2 style="font-size: 8pt"><tr>
<td><font face="verdana"><b>Fecha</b></font></td>
<td><font face="verdana"><b>Hora</b></font></td>
<td><font face="verdana"><b>Latitud</b></font></td>
<td><font face="verdana"><b>Longitud</b></font></td>
<td><font face="verdana"><b>Marcha</b></font></td>
<td><font face="verdana"><b>Velocidad</b></font></td>
<td><font face="verdana"><b>RPM</b></font></td>
<td><font face="verdana"><b>Luces</b></font></td>
</tr>

<?php
//Datos de la BD
  $link = @mysql_connect("localhost", "root", NULL)
```

```
    or die ("Error al conectar a la base de datos.");
//Tabla de la que se va a leer
 @mysql_select_db("agro", $link)
    or die ("Error al conectar a la base de datos.");
//Consulto a la BD los valores de posicion para asignar a variable y situar el mapa
 $query = "SELECT pos1, pos2 " .
   "FROM parametros ";
 $result = mysql_query($query);
 while($row = mysql_fetch_array($result))
 {
          $lati = $row['pos1'] .
          $lon = $row['pos2'];
 }
//Consulto a la BD todos los valores para rellenar la tabla
         $query = "SELECT pos1, pos2, rpm, luc, marcha, hora, fecha, vel " .
   "FROM parametros ";
 $result = mysql_query($query);
 while($row = mysql_fetch_array($result))
 {
         echo "<tr><td width=\"12.5%\"><font face=\"verdana\">" .
           $row["fecha"] . "</font></td>";
  echo "<td width=\"12.5%\"><font face=\"verdana\">" .
           $row["hora"] . "</font></td>";
  echo "<td width=\"12.5%\"><font face=\"verdana\">" .
           $row["pos1"] . "</font></td>";
  echo "<td width=\"12.5%\"><font face=\"verdana\">" .
           $row["pos2"]. "</font></td>";
```

```
            echo "<td width=\"12.5%\"><font face=\"verdana\">" .
                $row["marcha"] . "</font></td>";
    echo "<td width=\"12.5%\"><font face=\"verdana\">" .
                $row["vel"] . "</font></td>";
    echo "<td width=\"12.5%\"><font face=\"verdana\">" .
                $row["rpm"] . "</font></td>";
    echo "<td width=\"12.5%\"><font face=\"verdana\">" .
                $row["luc"]. "</font></td></tr>";
  }
//Libero el resultado y cierro la conexión con la tabla
  mysql_free_result($result);
  mysql_close($link);
?>
</table>


<p> </p>
<!--Comienza el script del mapa dado por la API de google-->
<div id="map"></div>
<script>
     function initMap() {
                        //Leo los valores de latitud y longitud del codigo PHP anterior
       var sitio = {lat: parseFloat('<?php echo $lati;?>'), lng: parseFloat('<?php echo $lon;?>')};
       var map = new google.maps.Map(document.getElementById('map'), {
         zoom: 4,
         center: sitio
       });
       var marker = new google.maps.Marker({
```

```
        position: sitio,
        map: map
      });
                    //Añado icono de tractor al marcdor
                    marker.setIcon('imgs/trac.png');
    }
  </script>
      <!--Incluyo mi clave de API de google para que funcione-->
  <script async defer
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBOvJ6CK7Q_GL7KAixluGZv4zIBTRr43cA&callback=initMap">
  </script>

</html>
```

9.-Código final Arduino

```
//Incluyo libreria para CAN y SPI
#include "mcp_can.h"
#include <SPI.h>

//Incluyo librerias para GPS y puerto serie software
#include <SoftwareSerial.h>
#include <TinyGPS.h>

//Defino pin del boton auxiliar
#define boton 5

//Defino objeto de tipo TinyGPS
TinyGPS gps;

//Defino puertos de serie software
SoftwareSerial neo6m(6, 7);
SoftwareSerial sim800l(3, 4);


//Variables para máquina de estados
int state = 0;
int nextstate = 0;
int enter = 0;    //Para antirebote, genera un pulso en esta
variable
int cuentae = 0;

//Variables necesarias para leer CAN
INT32U canId = 0x000;
unsigned char len = 0;
unsigned char buf[8];
int march = 0;
int luces = 0;
char str[20];
int alto = 0;
int bajo = 0;
const int SPI_CS_PIN = 9; //PIN CS
MCP_CAN CAN(SPI_CS_PIN);

//Variables para almacenar datos
int est_rpm = 0;
String est_marcha = "";
String est_luces = "";
long est_latitud = 0;
long est_longitud = 0;
unsigned long est_vel = 0;
unsigned long est_fecha;
unsigned long est_hora;

String urlbase ="\"83.53.13.58./datos.php?";
String bpos1 = "pos1=";      //Tramas base para añadir a la
trama principal con la IP
String bpos2 = "pos2=";
String bfecha = "fecha=";
```

```
String bhora = "hora=";
String bluc = "luc=";
String bmarcha = "marcha=";
String brpm = "rpm=";
String bvel = "vel=";

String pos1 = "";      //Tramas para añadir a la trama principal
con la IP y la trama con el dato que se envia
String pos2 = "";
String fecha = "";
String hora = "";
String luc = "";
String marcha = "";
String rpm = "";
String vel = "";

//Medida de tiempo
int tiem1 = 0;
int tiem2 = 0;
int tiem3 = 0;
int tiempociclo = 40000;  //Milisegundos que durara el ciclo
completo

//Variables auxiliares


//Variables necesarias para leer GPS
unsigned long fix_age, course;
unsigned long chars;
unsigned short sentences, failed_checksum;

void setup() {
  //Inicialización de puertos de serie soft, puerto hard para
depuracion
  Serial.begin(115200);
  while(!Serial);

    neo6m.begin(9600);  //El modulo GPS tiene una velocidad de
9600
    delay(100);
    Serial.println("Inicializado GPS y GPRS");

  //Inicialización comunicación SPI CAN
  START_INIT:
  if(CAN_OK == CAN.begin(CAN_500KBPS)){
        Serial.println("Inicializada comunicacion CAN BUS");
    }
  else{
        Serial.println("Fallo al inicializar CAN BUS");
        Serial.println("Volver a inicializar...");
        delay(100);
        goto START_INIT;
    }
}

void loop()
```

```
{
  if (tiem1 == 0){
    tiem1 = millis();
  }

  tiem2 = millis();
  tiem3 = tiem2 - tiem1; //Tiempo desde que empezó

switch(state){    //Máquina de estados
  case 0:
      if(nextstate == 0){  //Aprovecho la variable para escribir
solo una vez en el puerto de serie
          Serial.println("Leyendo GPS");
          nextstate = 1;
        }
      if(tiem3 > 2000){
          state = nextstate;
        }
      while (neo6m.available()){ //Si esta disponible el módulo
      int c = neo6m.read();
        if (gps.encode(c)){
          gps.get_position(&est_latitud, &est_longitud,
&fix_age); //Obtengo latitud y longitud en centesimas de grado
          gps.get_datetime(&est_fecha, &est_hora, &fix_age);
//Obtengo fecha y hora
          est_vel = gps.speed()*1.852/100;    //Obtengo
velocidad en km/h
          delay(100);   //Retardo para pasar al siguiente estado
        }
      }
    break;

  case 1:
      if(nextstate == 1){  //Aprovecho la variable para escribir
solo una vez
          neo6m.end();  //Cierro comunicacion GPS
        //Muestro los datos que he guardado del estado anterior
          Serial.println("Datos obtenidos GPS:");
          Serial.print("Latitud: ");
          Serial.println(est_latitud);
          Serial.print("Longitud: ");
          Serial.println(est_longitud);
          Serial.print("Velocidad: ");
          Serial.print(est_vel);
          Serial.println(" Km/h");
          Serial.print("Hora: ");
          Serial.print(est_hora);
          Serial.print("  Fecha: ");
          Serial.println(est_fecha);
          delay(1500);
          Serial.println("Leyendo bus CAN");
          nextstate = 2;
        }
      if(tiem3 > 6000){ //Leere el bus can durante 4 segundos
(6-2)
          state = nextstate;
```

```
        }
      if(CAN_MSGAVAIL == CAN.checkReceive()){
        CAN.readMsgBuf(&len, buf);
        canId = CAN.getCanId();

        if (canId == 217056256){
          for(int i = 0; i<len; i++){
            if (i == 3){
              bajo = int(buf[i]); //Guardo byte 3
            }
            if (i == 4){
              alto = int(buf[i]);   //Guardo byte 4
            }
          }
        }

        if (canId == 419372311){
          for(int i = 0; i<len; i++){
            if (i == 3){
              luces = int(buf[i]); //Guardo el byte 4 que
contiene la informacion de luces
            }
          }
        }

        if (canId == 217055747){
          for(int i = 0; i<len; i++){
            if (i == 0){
              march = int(buf[i]); //Guardo el byte 0 que
contiene la informacion de marcha/parado
            }
          }
        }

    }
    est_rpm = (bajo+(alto*256))/8;

    if ((march & B00000001) != 0){  //Mascara para comprobar si
el bit esta activo o no
      est_marcha = "SI";
    }
    else{
      est_marcha = "NO";
    }

    if ((luces & B00010000) != 0){  //Mascara para comprobar si
el bit esta activo o no
      est_luces = "SI";
    }
    else{
      est_luces = "NO";
    }

    break;

  case 2:
```

```
      if(nextstate == 2){  //Estado para construir las cadenas
de caracteres
        Serial.println("Datos obtenidos bus CAN");
        Serial.println("Luces: " + est_luces);
        Serial.println("Revoluciones por minuto: " +
String(est_rpm));
        Serial.println("Marcha: " + est_marcha);
        delay(200);
        Serial.println("Construyendo tramas para enviar");
        nextstate = 3;
      }
      fecha = bfecha + "'" + String(est_fecha).substring(0,2) +
"/" + String(est_fecha).substring(2,4) + "/" +
String(est_fecha).substring(4,6) + "'";
      hora = bhora + "'" + String(est_hora).substring(0,2) + ":"
+ String(est_hora).substring(2,4) + ":" +
String(est_hora).substring(4,6) + "'";
      pos1 = bpos1 + String(est_latitud).substring(0,
(String(est_latitud).length()-6)) + "." +
String(est_latitud).substring((String(est_latitud).length()-
6),String(est_latitud).length());
      pos2 = bpos2 + String(est_longitud).substring(0,
(String(est_longitud).length()-6)) + "." +
String(est_longitud).substring((String(est_longitud).length()-
6),String(est_longitud).length());
      vel = bvel + String(est_vel);
      luc = bluc + "'" + est_luces + "'";
      marcha = bmarcha + "'" + est_marcha + "'";
      rpm = brpm + String(est_rpm);
      delay(100);
      Serial.println(fecha);
      Serial.println(hora);
      Serial.println(pos1);
      Serial.println(pos2);
      Serial.println(vel);
      Serial.println(luc);
      Serial.println(marcha);
      Serial.println(rpm);
      state = nextstate;
    break;

  case 3:
      if(nextstate == 3){  //Aprovecho la variable para escribir
solo una vez
          Serial.println("Enviando informacion");
          nextstate = 4;
          sim800l.begin(113000);//El modulo GSM tiene una
velocidad de 113000
          delay(100);
        }
        enviadatos(pos1);
        delay(1000);
        enviadatos(pos2);
        delay(1000);
        enviadatos(fecha);
        delay(1000);
```

```
        enviadatos(hora);
        delay(1000);
        enviadatos(vel);
        delay(1000);
        enviadatos(luc);
        delay(1000);
        enviadatos(marcha);
        delay(1000);
        enviadatos(rpm);
        delay(1000);
        state = nextstate;
    break;

  case 4:
      if(nextstate == 4){  //Estado de espera
          Serial.println("Informacion enviada");
          nextstate = 0;
          sim800l.end();  //Cerrar comunicacion GSM
          delay(100);
          neo6m.begin(9600);
        }
        Serial.print("Siguiente lectura en ");
        Serial.print((tiempociclo-tiem3)/1000);
        Serial.println(" segundos");
      if(tiem3 > tiempociclo){
          state = nextstate;
          tiem1 = 0;
          tiem2 = 0;
        }
    break;

  case 5:   //Estado de diagnostico, se entra con el botón,
permite escribir comandos AT
      if(nextstate == 5){  //Aprovecho la variable para escribir
solo una vez
          Serial.println("Prueba modulo SIM800l");
          Serial.println("Esperando comandos AT");
          nextstate = 6;
          neo6m.end();
          sim800l.begin(113000);
        }
      //Programa puente entre puerto serie hardware y software
      while (sim800l.available()){ //Si esta disponible el
módulo
        Serial.write(sim800l.read());
      }
      while (Serial.available()){ //Enviar al sim800l lo que se
lea del puerto serie hardware
        sim800l.write(Serial.read());
      }
    break;

  case 6:   //Estado de diagnostico, se entra con el botón,
permite ver lo que envia el GPS bajo el estandar NMEA
      if(nextstate == 6){  //Aprovecho la variable para escribir
solo una vez
```

```
          Serial.println("Prueba modulo GPS");
          delay(1000); //Retardo para poder ver el mensaje
          sim800l.end();
          neo6m.begin(9600);
          nextstate = 7;
        }
      //Programa para ver el GPS en el puerto serie hardware
      while (neo6m.available()){ //Si esta disponible el módulo
        Serial.write(neo6m.read());
      }
    break;


  case 7:   //Estado de diagnostico, se entra con el botón,
permite ver la información de la línea CAN en bruto
      if(nextstate == 7){  //Aprovecho la variable para escribir
solo una vez
          Serial.println("Prueba modulo CAN");
          delay(1000); //Retardo para poder ver el mensaje
          nextstate = 0;
          tiem1 = 0;    //Cuando salga del estado podre volver a
empezar a contar tiempo
          tiem2 = 0;
        }
      if(CAN_MSGAVAIL == CAN.checkReceive()){
        CAN.readMsgBuf(&len, buf);
        canId = CAN.getCanId();
        Serial.print(canId);
        Serial.print(",");
           for(int i = 0; i<len; i++){
              Serial.print(buf[i]);
              Serial.print(",");
           }
           Serial.println();
        }
    break;
}

//Antirebote para el boton
  if (digitalRead(boton) == HIGH){
    if (cuentae < 10){
        cuentae = cuentae + 1;
        enter = 0;
      }
    else{
        if (cuentae == 21){           //Evitamos que mientras
sigamos pulsando siga avanzando estados
          enter = 0;
          }
        else{
          enter = 1;
          cuentae = 21;
          }
      }
    }
  else{
```

```
      cuentae = 0;
      enter = 0;
    }

//Acciones con el pulso que genera el antirebote al pulsar boton
  if (enter == 1){
    if (state < 5){
      nextstate = 5;
      state = 5;
      }
    else{
      state = nextstate;
    }
  }
}

void enviadatos(String datos){
  delay(100);
  sim800l.println("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");
  delay(100);
  leesim();
  delay(100);
  sim800l.println("AT+SAPBR=3,1,\"APN\",\"internet\"");
  delay(100);
  leesim();
  delay(100);
  sim800l.println("AT+SAPBR=1,1");
  delay(100);
  leesim();
  delay(100);
  sim800l.println("AT+HTTPINIT");
  delay(100);
  leesim();
  delay(100);
  sim800l.println(("AT+HTTPPARA=\"URL\",") + urlbase + datos +
"\""); //Añado comillas para terminar la orden
 /*
sim800l.println("AT+HTTPPARA=\"URL\",\"83.53.13.58./datos_2.php?
hora=\"");*/
  delay(2000);
  leesim();
  delay(100);
  sim800l.println("AT+HTTPACTION=0");
  delay(2000);
  leesim();
  delay(100);
}

void leesim(){
  while (sim800l.available()){ //Si esta disponible el módulo
        Serial.write(sim800l.read());
      }
  }
```

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Gestión telemática de vehículos agrícolas

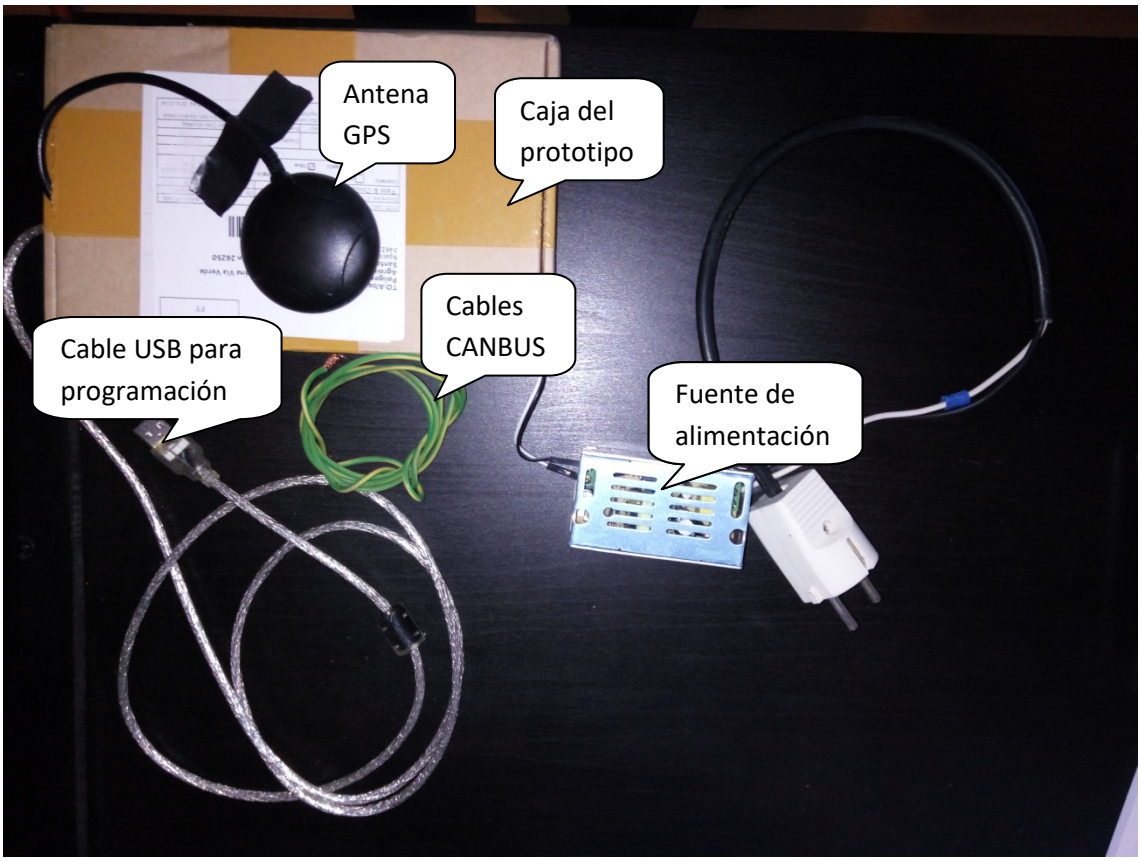
DOCUMENTO 4: ANEXO II IMÁGENES DEL MONTAJE

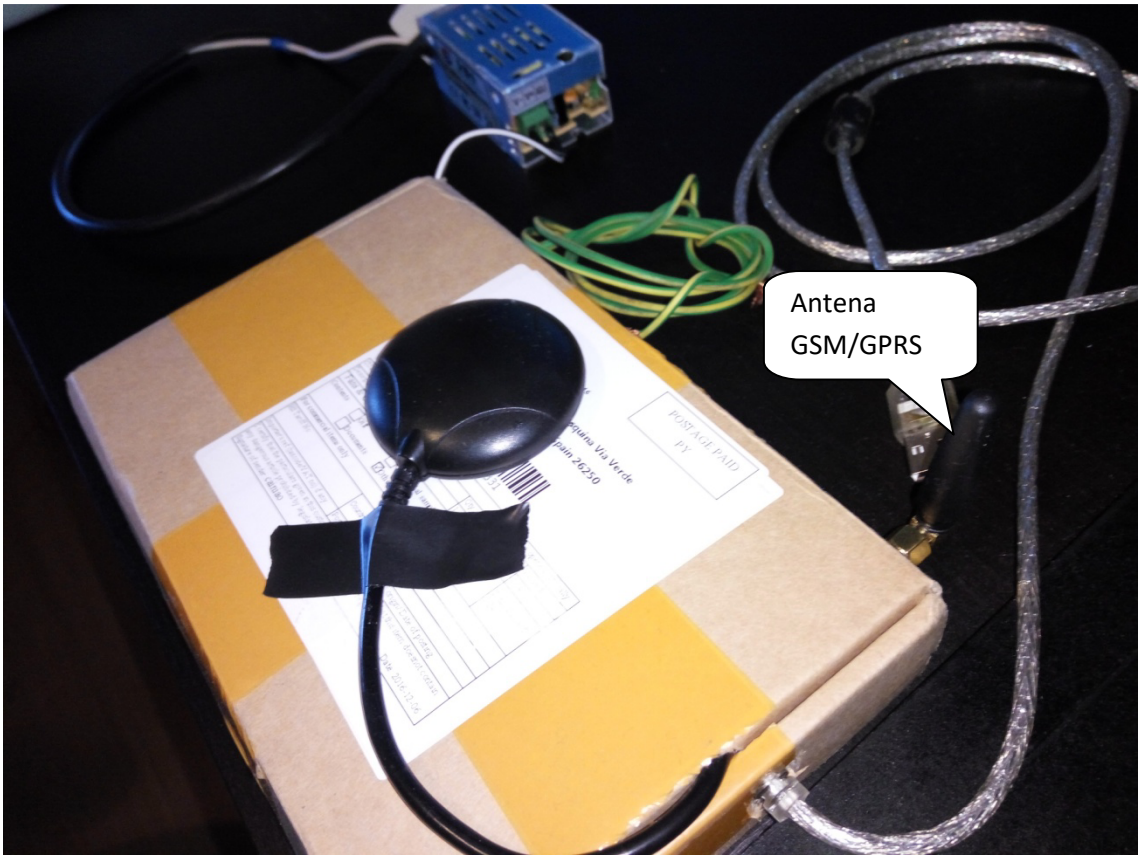
Alberto Úbeda Cañas

Marzo de 2017

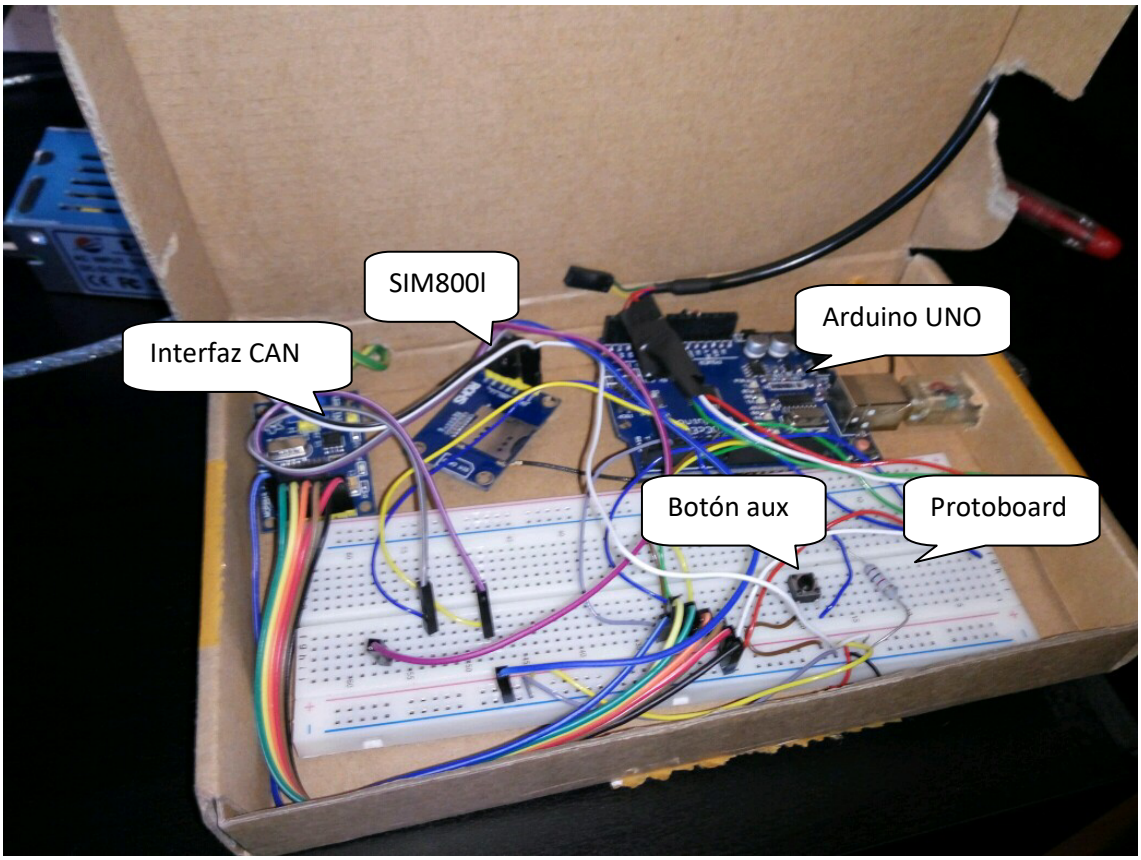
ANEXO II: IMÁGENES DE MONTAJE

1.-Montaje general





2.-Interior de la caja



3.-Alimentación

Toda la alimentación empleada en el prototipo proviene de una fuente de alimentación comercial que suministra 5 Voltios y 2 Amperios a partir de la red de alimentación de 220V de corriente alterna.

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Gestión telemática de vehículos agrícolas

DOCUMENTO 5: PLANOS

Alberto Úbeda Cañas

Marzo de 2017

PLANOS

1.-Convertidor 12V – 5V 3 Amperios

Con el software KiCAD se ha diseñado el circuito para convertir la corriente de 12V que viene de la batería del vehículo a 5V.

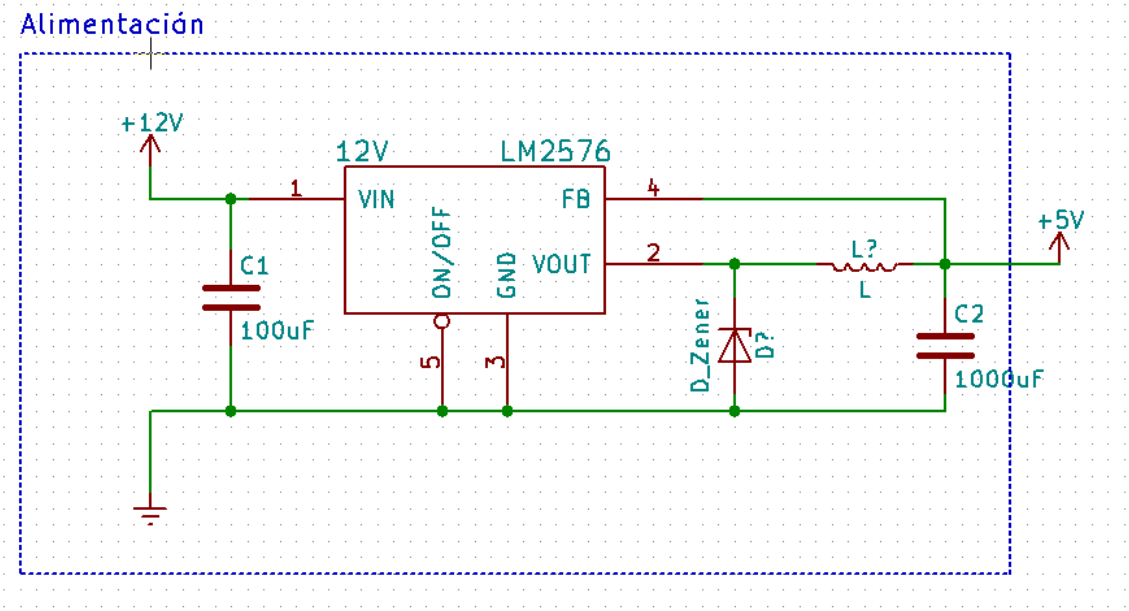
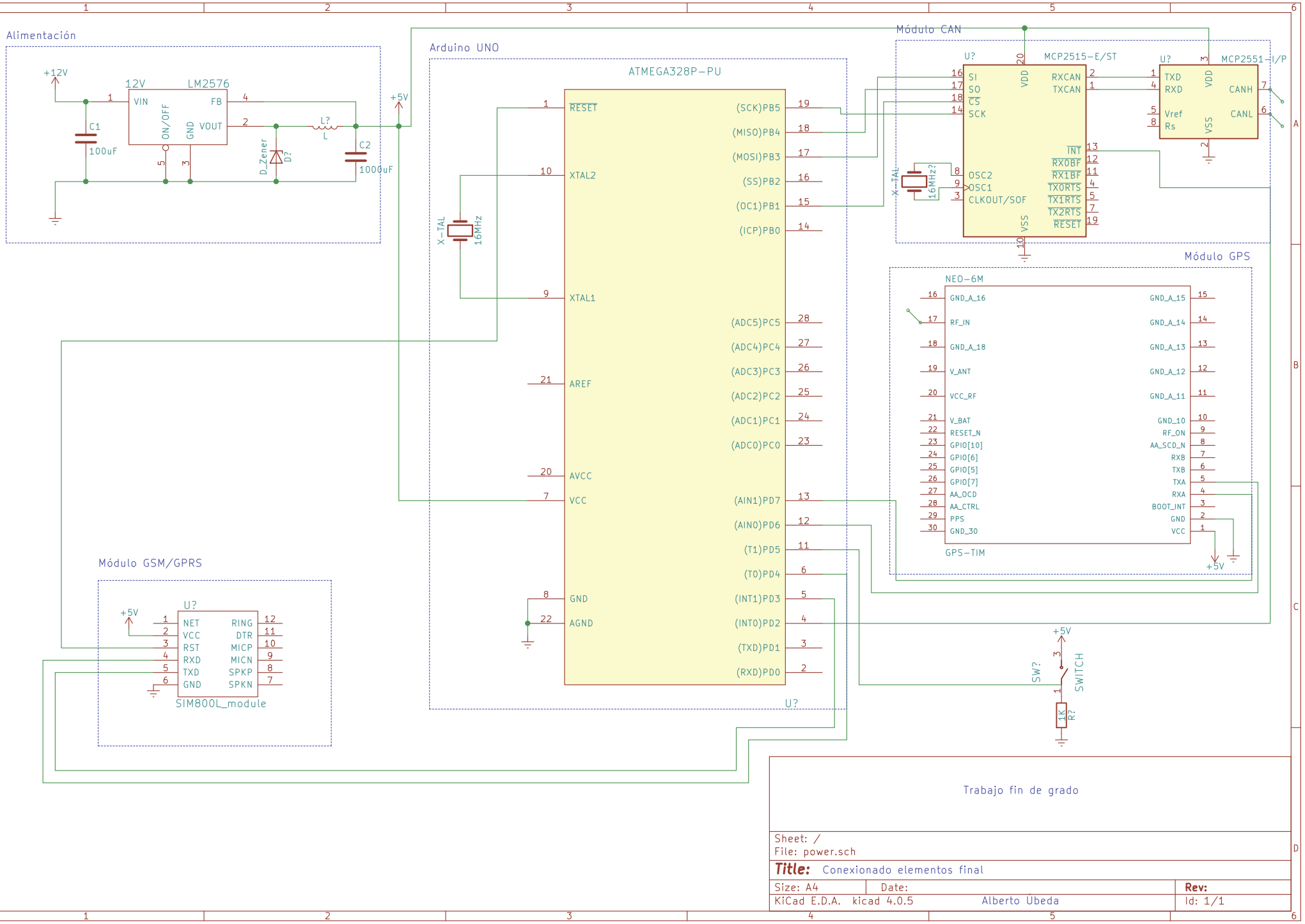


Ilustración 1: Plano módulo alimentación


Alimentación
+12V
12V
LM2576
VIN
ON/OFF
GND
FB
VOUT
C1
100uF
L?
L
D_Zener
D?
C2
1000uF
+5V
Arduino UNO
ATMEGA328P-PU
RESET
XTAL2
XTAL1
AREF
AVCC
VCC
GND
AGND
(SCK)PB5
(MISO)PB4
(MOSI)PB3
(SS)PB2
(OC1)PB1
(ICP)PB0
(ADC5)PC5
(ADC4)PC4
(ADC3)PC3
(ADC2)PC2
(ADC1)PC1
(ADC0)PC0
(AIN1)PD7
(AIN0)PD6
(T1)PD5
(T0)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
X-TAL
16MHz
U?
Módulo CAN
MCP2515-E/ST
SI
SO
CS
SCK
VDD
RXCAN
TXCAN
INT
RX0BF
RX1BF
TX0RTS
TX1RTS
TX2RTS
RESET
OSC2
OSC1
CLKOUT/SOF
VSS
X-TAL
16MHz?
MCP2551-I/P
TXD
RXD
Vref
Rs
CANH
CANL
Módulo GPS
NEO-6M
GND_A_16
RF_IN
GND_A_18
V_ANT
VCC_RF
V_BAT
RESET_N
GPIO[10]
GPIO[6]
GPIO[5]
GPIO[7]
AA_OCD
AA_CTRL
PPS
GND_30
GND_A_15
GND_A_14
GND_A_13
GND_A_12
GND_A_11
GND_10
RF_ON
AA_SCD_N
RXB
TXB
TXA
RXA
BOOT_INT
GND
VCC
GPS-TIM
Módulo GSM/GPRS
NET
VCC
RST
RXD
TXD
GND
RING
DTR
MICP
MICN
SPKP
SPKN
SIM800L_module
SW?
SWITCH
1K
R?
Trabajo fin de grado
Sheet: /
File: power.sch
Title: Conexionado elementos final
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.5
Alberto Úbeda
Id: 1/1

3.-Vista 3D Caja

Vista tapa

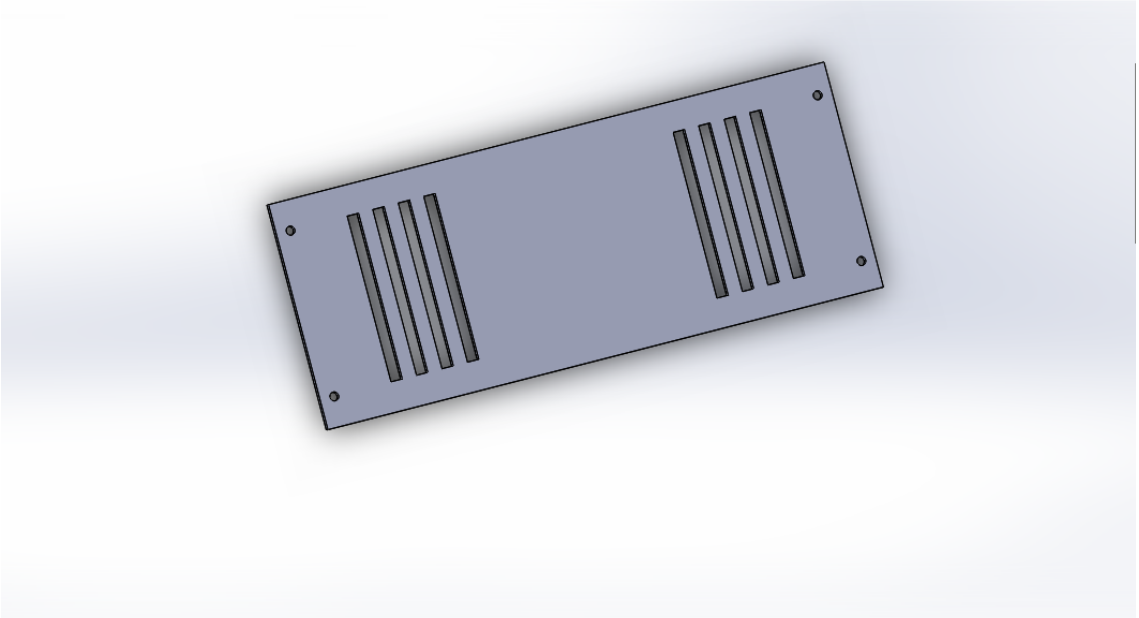


Ilustración 2: Vista 3D de la tapa de la caja

Vista cuerpo

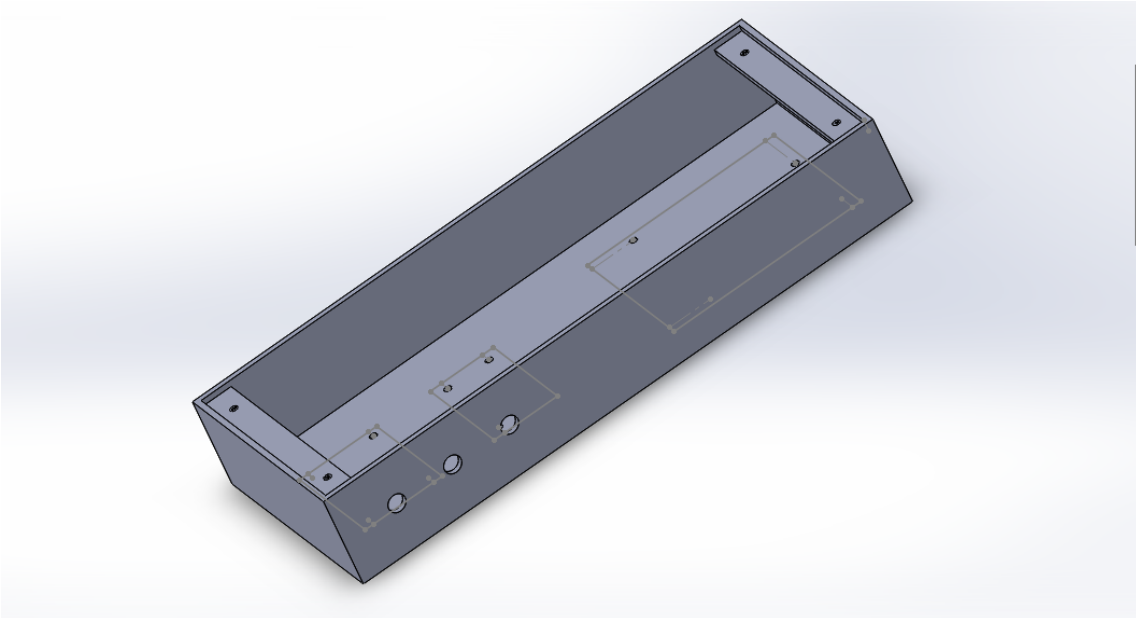
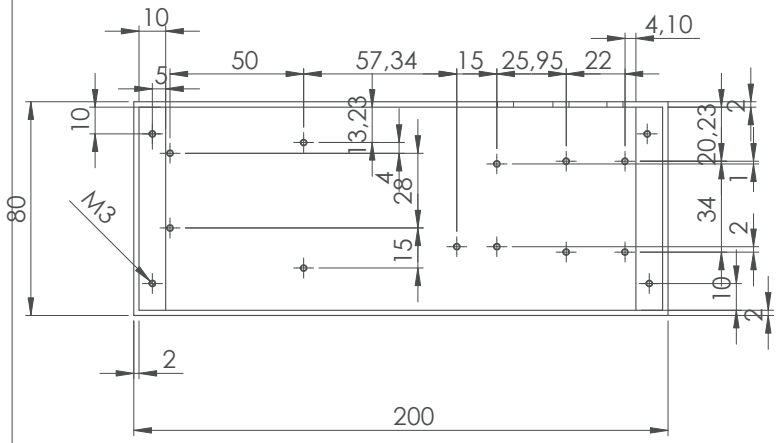
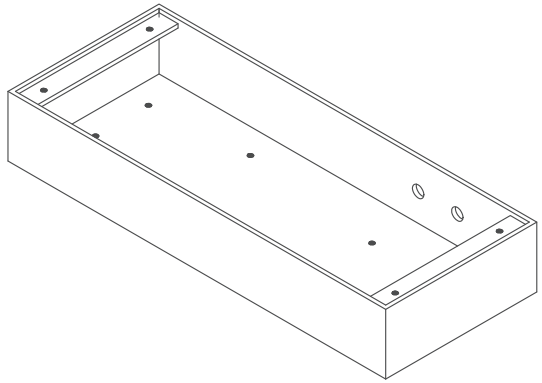
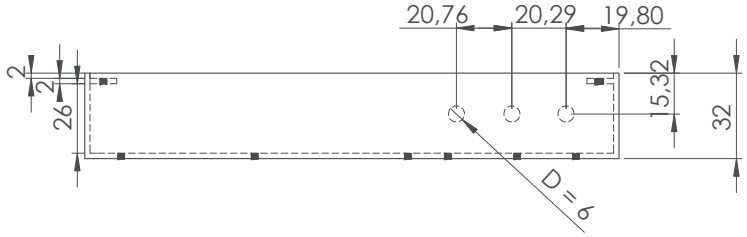


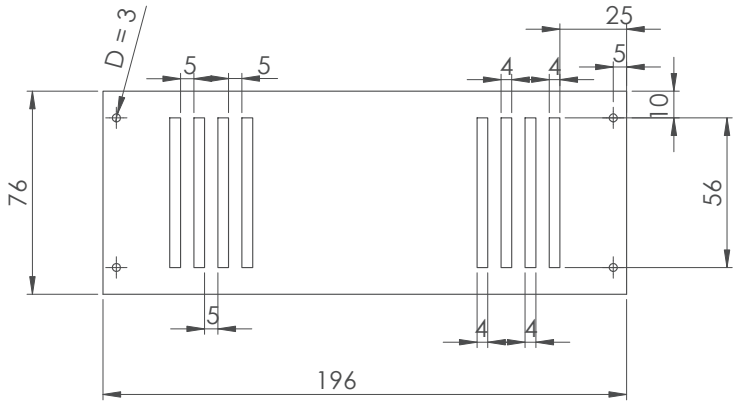
Ilustración 3: Vista 3D de la caja del prototipo

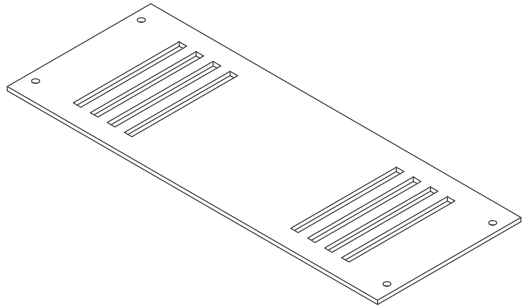






Cantidad	Denominación		Marca	Plano Nº	Modelo	Material	Peso	Observaciones
	FECHA	NOMBRE						
Dibujado		Alberto						
Comprobado								
Id.s.normas								
ESCALAS 1:2	Caja prototipo (Cuerpo)						Número	
PROYECCIÓN	Trabajo Fin de Grado						REFERENCIA	
							Sustituye a:	
							Suistituido por:	





Cantidad	Denominación		Marca	Plano Nº	Modelo	Material	Peso	Observaciones
	FECHA	NOMBRE						
Dibujado		Alberto						
Comprobado								
Id.s.normas								
ESCALAS 1:2	Caja prototipo (tapa)					Número		
PROYECCIÓN	Trabajo fin de grado					REFERENCIA		
						Sustituye a:		
						Suistituido por:		

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Gestión telemática de vehículos agrícolas

DOCUMENTO 6: PLIEGO DE CONDICIONES

Alberto Úbeda Cañas

Marzo de 2017

PLIEGO DE CONDICIONES

1.-Consideraciones legales

En este primer apartado se indicarán las disposiciones legales y normas técnicas que deberán ser tomadas en cuenta para la realización del presente proyecto.

Es importante entonces definir la diferencia entre norma técnica y ley, como aspectos más relevantes se dirá que una ley es de obligado cumplimiento y es dictada por el estado, o en su caso por la Unión Europea; mientras que las normas tienen un carácter de cumplimiento voluntario, son fruto de un consenso alcanzado tras la experiencia y los avances tecnológicos y son editadas por un organismo de normalización reconocido (AENOR en el caso de España).

En el ámbito legal, la única ley que afecta directamente a este proyecto es la "Ley 9/2014, de 9 de mayo, General de Telecomunicaciones". Puesto que el presente sistema utiliza comunicaciones GSM/GPRS es muy importante que bajo ningún concepto se vulnere la mencionada ley.

El sistema de comunicación GPS cumplirá los estándares establecidos por la norma UNE-EN 61108 con título "Equipos y sistemas de navegación marítima y radiocomunicación".

Se cumplirá también con los estándares establecidos en las normas ISO11898 (bus CAN) e ISO11783 (ISOBUS), que definen las comunicaciones internas del vehículo, y por lo tanto como se comunicará este con el sistema desarrollado.

Es deber del fabricante asegurarse que todo el sistema desarrollado (Tanto hardware como software) cumple con las leyes y normativa vigentes en el momento del desarrollo.

2.-Consideraciones técnicas

En este apartado se van a describir los medios materiales y las características de la tecnología a utilizar

Prototipo

Para la construcción del prototipo, se verificará que todos los componentes y módulos empleados cumplen en estándar CE de acuerdo con los principios de la Decisión 768/2008/CE del Parlamento Europeo y del Consejo de fecha 9 de julio de 2008, que certifica que un producto cumple con los estándares de seguridad y salud exigidos dentro de la Unión Europea.

Los cables empleados para alimentación, y para conexión del sistema a la línea CAN tendrán una sección de 1.5mm^2. En el caso de los cables de alimentación se identificarán con los colores rojo y negro para positivo y negativo respectivamente; mientras que para los cables de conexión a la línea CAN se emplearán colores diferentes a los anteriores con objetivo de diferenciarlos. Además estos cables se etiquetarán con las marcas “CAN-H” para la conexión CAN alto y “CAN-L” para la conexión CAN bajo.

Los cables tipo Dupont empleados para las conexiones electrónicas tendrán una longitud máxima de 20 centímetros. Además cumplirán la norma de calibre de alambre estadounidense AWG22 (0.326mm^2 de área).

Caja de prototipo

Se fabricará en aluminio, con objetivo de favorecer la refrigeración del sistema. En el proceso de construcción podrá emplearse una plancha de aluminio de 2 milímetros de espesor, realizando los pliegues que sean necesarios. Sobre este material base se mecanizarán posteriormente las roscas y agujeros que sean necesarios. También se realizarán soldaduras si es necesario.

Cabe la opción también de fabricar la caja entera mediante control numérico. Se partirá en ese caso de un tocho de aluminio de dimensiones suficientes para poder construir toda la pieza.

Servidor Web

El equipo empleado como servidor web deberá cumplir unos requisitos mínimos de hardware: Al menos 4 Gigabytes de memoria RAM, almacenamiento de al menos 50 Gigabytes y tarjeta de red con conectividad hasta 1000Mbps, asegurando capacidad suficiente para poder enviar y recibir los datos sin ningún problema incluso en caso de que se realicen ampliaciones en el sistema.

3.-Consideraciones de distribución y ampliación

El fabricante del sistema no podrá desarrollar sistemas similares al aquí descrito para otros clientes durante un plazo de 2 años a contar desde la entrega del producto.

Asimismo, el fabricante podrá ser solicitado por el cliente para la fabricación de más sistemas o para la inclusión de mejoras en los ya fabricados, abonando el cliente los gastos derivados de todas estas modificaciones. Se estipula un tiempo en el que el fabricante tendrá obligación de dar este soporte al cliente de 4 años a contar desde la entrega del producto.

El cliente no tiene permitida la venta o distribución del producto derivado del presente proyecto a terceras personas sin consentimiento del fabricante, por lo que, según lo estipulado en el principio de este apartado, durante dos años el producto solo podrá ser comercializado entre las dos partes que forman parte de este trabajo (fabricante y cliente).

4.-Consideraciones de instalación y uso

Equipo instalado en el vehículo

El usuario no deberá tener ningún tipo de problema con respecto al equipo instalado en su vehículo, por lo que el manejo e interacción con el vehículo será exactamente el mismo que antes de la instalación. El equipo trabajará de manera autónoma y totalmente transparente. Los datos se enviarán sin que el usuario reciba ningún tipo de aviso o notificación.

La instalación del equipo será realizada por un técnico profesional en un lugar adecuado, como podría ser un taller de reparaciones mecánicas y eléctricas.

La instalación se realizará de manera que todo el sistema sea transparente al usuario del vehículo, es decir, el sistema inicializará su funcionamiento cuando lo haga el vehículo, y lo cesará de igual manera.

Puesto que el sistema no cuenta con un sistema de aviso acústico o sonoro de fallos, estos se detectarán cuando haya problemas con la recepción de los datos en el servidor. En este caso, el usuario contactará con la institución que realizó la instalación del sistema, pudiendo esta realizar un diagnostico rápido del equipo mediante la conexión USB que este posee. En caso de errores no subsanables por el personal técnico, se contactará con el fabricante del equipo.

El fabricante se reserva el derecho de desarrollar actualizaciones de software para el sistema, en caso de ocurrir se informará a los clientes finales, de manera que estos, voluntariamente puedan acudir al lugar donde se realizó la instalación para obtener las últimas mejoras del software del dispositivo.

Asimismo, el instalador tendrá la obligación de dar una garantía de dos años por su trabajo, haciéndose cargo de todos los fallos y problemas derivados de las acciones realizadas.

Los posibles fallos o averías derivados de vicios o del funcionamiento normal del equipo, serán cubiertos por una garantía de dos años otorgada por el fabricante.

Servidor Web

La instalación del servidor web para almacenaje y consulta de datos será llevada a cabo por el fabricante del equipo. Además también realizará todas las actividades de configuración y puesta en marcha necesarias.

El mantenimiento y reparaciones sencillas del servidor será subcontratado a una tercera empresa, a elegir por el cliente. En caso de ocurrir algún fallo o avería no subsanable por el subcontratista, se contactará con el equipo de servicio técnico del fabricante.

El fabricante proporcionará dos años de garantía sobre el servidor, cubriendo los fallos derivados del funcionamiento normal de este. Nunca de los fallos derivados de una mala utilización (averías por agua, golpes, etc)

5.-Consideraciones económicas

El presupuesto asciende a 6368.59€ IVA incluido, el mismo se desglosa en:

- Partida 1: Estudio de la solución.
- Partida 2: Desarrollo del prototipo.
- Partida 3: Desarrollo del servidor web.

Será obligación del cliente aportar el pago correspondiente a la denominada como “Partida 1” en los 15 días posteriores a la firma del contrato de inicio del proyecto.

Asimismo deberá abonar la cantidad correspondiente a la “Partida 2” una vez concluya el estudio de la solución y comience el desarrollo del prototipo.

La partida 3 será abonada en el momento de la entrega de la solución final, dándose por finalizado el proyecto y comenzando a contar un plazo de garantía de dos años a partir de ese momento. Como se ha mencionado en apartados anteriores, dicha garantía se hará cargo de los fallos y averías derivados del propio funcionamiento del sistema, además de las posibles afecciones que haya habido hacia otros elementos. La garantía nunca se hará cargo de averías derivadas del mal uso del sistema, estas correrán siempre a cargo del cliente.

Firmado por: Fecha:

DNI:

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Gestión telemática de vehículos agrícolas

DOCUMENTO 7: PRESUPUESTO

Alberto Úbeda Cañas

Marzo de 2017

PRESUPUESTO

1.-Listado de precios unitarios

Código	Ud	Descripción	Precio (€)
TFGPP001	h	Técnico instalador	18.90
TFGPP002	h	Ingeniero electrónico	28.60
TFGPP003	u	Arduino Uno	23.45
TFGPP004	u	Protoboard	6.51
TFGPP005	u	Módulo MCP2515/2551	8.24
TFGPP006	u	SIM800l GSM/GPRS	15.25
TFGPP007	u	NEO-6M GPS	17.14
TFGPP008	u	Cables dupont	0.10
TFGPP009	u	Servidor Web HP Microserver G8	222.95
TFGPP010	u	Caja de montaje	42.35
TFGPP011	u	Tornillo métrica 3 x 5mm	0.03
TFGPP012	u	Separador hexagonal métrica 3 x 5mm	0.14
TFGPP013	h	Instalador autorizado	21.40
TFGPP014	h	Programador	23.50
TFGPP015	h	Montaje del sistema	62.10

2.-Listado de precios descompuestos

Capítulo I: Estudio de la solución

Código	Cantidad (Uds)	Descripción	Precio (€)	Subtotal (€)	Importe (€)
TFGPP007	1 u	NEO-6M GPS	17.14	17.14	
TFGPP003	1 u	Arduino Uno	23.45	23.45	
TFGPP004	1 u	Protoboard	6.51	6.51	
TFGPP005	1 u	Módulo MCP2515/2551	8.24	8.24	
TFGPP006	1 u	SIM800l GSM/GPRS	15.25	15.25	
TFGPP008	40 u	Cables dupont	0.10	4.00	
TFGPP002	45 h	Ingeniero electrónico	28.60	1287.00	

Suma la partida..... 1361.59€

Costes indirectos (4%)..... 54.46€

Total partida..... 1416.05€

Capítulo II: Desarrollo de prototipo

Código	Cantidad (Uds)	Descripción	Precio (€)	Subtotal (€)	Importe (€)
TFGPP010	1 u	Caja de prototipo	42.35	42.35	
TFGPP002	80 h	Ingeniero electrónico (diseño electrónico)	28.60	2288.00	
TFGPP002	10 h	Ingeniero electrónico (diseño caja prototipo)	28.60	286.00	
TFGPP011	15 u	Tornillo métrica 3 x 5mm	0.03	0.45	
TFGPP012	11 u	Separador métrica 3 x 5mm	0.14	1.54	

Suma la partida..... 2618.34€

Costes indirectos (4%)..... 104.73€

Total partida..... 2723.07€

Capítulo III: Desarrollo e instalación de servidor web

Código	Cantidad (Uds)	Descripción	Precio (€)	Subtotal (€)	Importe (€)
TFGPP009	1 u	Servidor Web HP Microserver G8	222.95	222.95	
TFGPP002	30 h	Ingeniero electrónico (Desarrollo servidor web)	28.60	858.00	

Suma la partida..... 1080.95€

Costes indirectos (4%)..... 43.23€

Total partida..... 1124.18€

3.-Resumen del presupuesto del proyecto

Capítulo	Resumen	Importe (€)
Capítulo I	Estudio de la solución	1416.05
Capítulo II	Desarrollo del prototipo	2723.07
Capítulo III	Desarrollo e instalación de servidor web	1124.18

Total ejecución.....5263.30

IVA 21%.....1105.29

TOTAL PRESUPUESTO CONTRATO.....6368.59

El precio total del proyecto asciende a la cantidad de SEIS MIL TRESCIENTOS SESENTA Y OCHO EUROS CON CINCUENTA Y NUEVE CÉNTIMOS.

4.-Presupuesto por unidad fabricada e instalada.

En este capítulo se especifica el precio que tendrá cada unidad que el comprador adquiera e instale una vez el sistema haya sido desarrollado por completo.

Código	Cantidad (Uds)	Descripción	Precio (€)	Subtotal (€)	Importe (€)
TFGPP007	1 u	NEO-6M GPS	17.14	17.14	
TFGPP003	1 u	Arduino Uno	23.45	23.45	
TFGPP004	1 u	Protoboard	6.51	6.51	
TFGPP005	1 u	Módulo MCP2515/2551	8.24	8.24	
TFGPP006	1 u	SIM800l GSM/GPRS	15.25	15.25	
TFGPP008	40 u	Cables dupont	0.10	4.00	
TFGPP015	1 h	Montaje del sistema	62.10	62.10	
TFGPP013	2 h	Instalación en vehículo	21.40	42.80	
TFGPP014	1 h	Programación del sistema	23.50	23.50	
TFGPP010	1 u	Caja de montaje	42.35	42.35	
				Total.....	245.35€
				IVA (21%).....	51.52€
				Total unidad.....	**296.87€**

El precio unitario asciende a DOSCIENTOS NOVENTA Y SEIS EUROS CON OCHENTA Y SIETE CENTIMOS.